
Alerta Documentation

Release 8.2

Nick Satterly

Nov 25, 2021

CONTENTS

1	Demo Sites	3
1.1	Quickstart	3
1.2	Design Principles	4
1.3	Server & API	5
1.4	Alerta Web UI	9
1.5	Alerta CLI	9
1.6	Integrations & Plugins	23
1.7	Configuration	29
1.8	Authentication	42
1.9	Authorization	53
1.10	Deployment	57
1.11	Plug-ins	61
1.12	Custom Webhooks	62
1.13	Customer Views	62
1.14	Federated Alerta	63
1.15	Conventions	66
1.16	Development	68
1.17	Getting Started	69
1.18	Resources	70
1.19	API Reference	71
1.20	API Query Syntax	113
1.21	Alert Format	118
1.22	Heartbeat Format	122
2	Contribute	125
3	Support	127
3.1	Frequently Asked Questions	127
4	License	131
4.1	Releases	131
4.2	About	137
5	Indices and tables	139
	Index	141

The alerta monitoring system is a tool used to consolidate and de-duplicate alerts from multiple sources for quick ‘at-a-glance’ visualisation. With just one system you can monitor alerts from many other monitoring tools on a single screen.

Severity	Status	Last Receive Time	Dupl.	Environment	Service	Resource	Event	Value	Text
Critical	Open	Fri 16 Oct 00:55	0	Production	Torniskel	Ockey	breachOfConfidentiality	n/a	Marfa Godard Banksy voluptate cred. Cray gastrop
Critical	Open	Fri 16 Oct 00:40	0	Production	Notestack	Extremes	InputOutputDeviceError	n/a	Aesthetic chia biodiesel, shabby chic Brooklyn Mar
Major	Open	Fri 16 Oct 00:56	0	Production	Deployinator	kue	dataSetOrModelError	n/a	Nihil crucifix tousled, shabby chic cardigan et me
Major	Open	Fri 16 Oct 00:40	0	Production	Support	stylebot	fileError	n/a	Commodo roof party gluten-free placeat. Forage qu
Major	Open	Fri 16 Oct 00:39	0	Production	Mactio	jslint.vim	cableTamper	n/a	Hella squid dolor readymade Banksy, master cleans
Minor	Open	Fri 16 Oct 00:55	0	Production	Firetray	spine.contacts	lanError	n/a	Laboris gastropub artisan occaecat, tattooed VHS
Minor	Open	Fri 16 Oct 00:39	0	Production	Phnix	cssSandpaper	communicationsProtocolError	n/a	Church-key veniam commodo, put a bird on it try-ha
Warning	Open	Fri 16 Oct 00:55	0	Production	Friar	Pxloader	communicationsSubsystemFailure	n/a	Odio minim laborum, cupidatat Shoreditch biodiesel
Warning	Open	Fri 16 Oct 00:55	0	Production	Yschool	bbb	adapterError	n/a	Sustainable PBR cardigan Brooklyn, listicle elit o
Indeterminate	Open	Fri 16 Oct 00:56	0	Production	Jsconfus	webglreport	adapterError	n/a	Neutra vegan tilde, proident keytar swag dreamcatc
Indeterminate	Open	Fri 16 Oct 00:55	0	Production	Photoswipe	RealTimeMonitor	dteDceInterfaceError	n/a	Roof party listicle cillum, enim Tumblr DIY artisa
Indeterminate	Open	Fri 16 Oct 00:55	0	Production	Wireit	currency.io	configurationOrCustomizationError	n/a	Biodiesel labore 8-bit, odio eu American Apparel
Indeterminate	Open	Fri 16 Oct 00:55	0	Production	Zia	openstreetbugs	proceduralError	n/a	Sustainable selvage slow-carb brunch bespoke odio
Indeterminate	Open	Fri 16 Oct 00:55	0	Production	Nodelint	Lightface	ouputDeviceError	n/a	Meh qui brunch, twee mixtape Shoreditch dolor Pi

Showing 14 out of 14 alerts

Alerta combines a JSON API *server* for receiving, processing and rendering alerts with a simple, yet effective *Alerta Web UI* and *command-line tool*. There are numerous *integrations* with popular monitoring tools and it is easy to add your own using the *API* directly, the *Python SDK* or the same command-line tool to *send alerts*. Access to the API and command-line tool can be restricted using *API keys* and to the web console using Basic Auth or OAuth2 providers Google, GitHub and GitLab.

Get started today!

DEMO SITES

There are two public web consoles available for demonstration and testing:

- <https://try.alerta.io> (Google OAuth)
- <https://alerta.herokuapp.com> (BasicAuth)

The web consoles are powered by a single public API which can be used as a sandbox for integration testing:

- <https://alerta-api.herokuapp.com>

The “API Explorer” can be used to query for and send alerts to the public API server:

- <https://explorer.alerta.io>

The alerta command-line tool can also be used to generate alerts. The required API key is `demo-key`.

1.1 Quickstart

This is a quick-start guide that will get you running Alerta in under 5 minutes.

1.1.1 Install MongoDB

For Debian/Ubuntu, run:

```
$ sudo apt-get install -y mongodb-org
$ sudo mongod -f /etc/mongod.conf &
```

If `apt-get` can’t locate the “mongodb-org” metapackage package then follow [these steps](#) to add MongoDB package repository to apt sources list.

For other operating systems, see the [installation](#) steps on the MongoDB web site.

1.1.2 Install the Alerta Server

To install the alerta server:

```
$ pip3 install alerta-server
$ alertad run --port 8080
```

You should see something like:

```
* Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
```

1.1.3 Install the Web Console

To install the web console:

```
$ wget https://github.com/alerta/alerta-webui/releases/latest/download/alerta-webui.tar.  
↪gz  
$ tar zxvf alerta-webui.tar.gz  
$ cd dist  
$ python3 -m http.server 8000  
  
>> browse to http://localhost:8000
```

1.1.4 Install the Alerta CLI

To install the alerta command-line tool:

```
$ pip3 install alerta
```

1.1.5 Send some alerts

To send an alert to the server:

```
$ alerta send -r web01 -e NodeDown -E Production -S Website -s major -t "Web server is  
↪down." -v ERROR
```

The alert should appear almost immediately in the console. If it doesn't it's either a CORS issues or a [bug](#).

1.1.6 What's next?

Take the step-by-step tutorials or dive straight into a [deployment](#).

1.2 Design Principles

The following principles guided the design and development of the Alerta monitoring system.

1.2.1 Resource under alarm

A *resource* is any entity that it makes sense for you to receive alerts for. You shouldn't be forced to accept a certain "world view" when using a monitoring tool or to repurpose a "host" field for a service or application, or a even a URL. Host-centric monitoring tools belong in the 90's.

1.2.2 Many severity levels

You are free to use as many or as few as you like eg. if you plan to only integrate with Nagios then only use `critical`, `warning` and `ok`. If you are integrating with a fault management system for a telco you might want to use the six [ISO perceived severity levels](#) or alternatively, if you are pushing application alerts you might want to consider using the `debug` and `trace` severity levels.

1.2.3 Robust alert reception

In accordance with the [robustness principle](#) which is to “be liberal in what you accept from others”, alerta will accept any alert as long as it meets the alert format specification. ie. no field values need to be defined in advance for it to be accepted, however the benefits of following a standard [convention](#) for such attributes as `environment`, `service`, `event` and `resource` (as internally defined by and useful to you) are many.

1.2.4 Self-clearing alerts

All alerts should have a corresponding `cleared` or `normal` state so that non-normal alerts can be automatically cleared down by the system. Where an alert cannot send a corresponding clear an alert should specify a `timeout` (or have a default assigned) after which it will be deleted.

1.2.5 Alerts are cheap

Alerts should be resent at regular intervals if they are still active which means that if all data is lost after a certain amount of time (eg. 2 hours?) you are back to where you were. This will be generally true though, for some alert sources this isn't possible eg. SNMP traps, log errors. Alerts in a normal state can be resent at a longer interval.

1.2.6 Tags and custom attributes

Dynamic ‘scale up’/‘scale down’ environments are the defacto standard now; naming individual servers is lame. Use service discovery and dynamically generated metadata to tag alerts and assign custom attributes on the fly.

1.3 Server & API

The Alerta API receives alerts from multiple sources, [correlates](#), [de-duplicates](#) or [suppresses](#) them, and makes the alerts available via a [RESTful JSON API](#).

Alerts can be intercepted as they are received to modify, enhance or reject them using [pre-receive hooks](#). Alerts can also be used to trigger actions in other systems after the alert has been processed using [post-receive hooks](#) or following an [operator action](#) or alert [status change](#) for bi-directional integration.

There are several [integrations](#) with popular monitoring tools available and [webhooks](#) can be used to trivially integrate with AWS Prometheus, Grafana, PagerDuty and many more.

1.3.1 Event Processing

Alerta comes *out-of-the-box* with key features designed to reduce the burden of alert management. When an event is received it is processed in the following way:

1. all plugin pre-receive hooks are run in listed order, an alert is immediately rejected if any plugins return a `RejectException` or `RateLimit` exception
2. alert is checked against any active blackout periods, alert suppressed if any match
3. alert is checked if duplicate, if so duplicate count is increased and repeat set to *True*
4. alert is checked if correlated, if so change severity and/or status etc
5. if alert is neither a duplicate or correlated then create new alert
6. all plugin post-receive hooks are run in listed order
7. any tags or attributes changed in post-receive hooks are persisted

Each of the above actions are explained in more detail in the following sections.

1.3.2 Plugins

Plugins are small python scripts that can run either before or after an alert is saved to the database, or before an operator action or status change update. This is achieved by registering *pre-receive hooks* for transformers, *post-receive hooks* for external notification and *status change hooks* for bi-directional integration.

Transformers

Using pre-receive hooks, plugins provide the ability to transform raw alert data from sources before alerts are created. For example, alerts can be *normalised* to ensure they all have specific attributes or tags or only have a specific value from a range of allowed values. This is demonstrated in the [reject plugin](#) that enforces an alert policy.

Plugins can also be used to *enhance* alerts – like the [Geo location plugin](#) which adds location data to alerts based on the remote IP address of the client, or the generic [enhance plugin](#) which adds a customer attribute based on information contained in the alert.

External Notification

Using post-receive hooks, plugin integrations can be used to provide downstream systems with alerts in realtime for external notification. For example, pushing alerts onto an [AWS SNS topic](#), [AMQP queue](#), logging to a [Logstash/Kibana](#) stack, or sending notifications to [HipChat](#), [Slack](#) or [Twilio](#) and many more.

Operator Actions

Actions taken against alerts can be used as triggers for further integrations with external systems.

TBC

Bi-directional Integration

Using status change hooks, plugins can be used to complete a two way integration with an external system. That is, an external system like Prometheus Alertmanager that generates alerts that are forwarded to Alerta can be updated when the status of an alert changes in Alerta.

For example, if an operator “acknowledges” a Prometheus alert in the Alerta web UI then a status change hook could silence the corresponding alert in Alertmanager. This requires that external systems provide enough information in the alert created in Alerta for that alert to be uniquely identified at a later date.

More information about bi-directional integration and real-world examples for Telegram, Zabbix, Prometheus and many others can be found on the [Integrations & Plugins](#) page.

1.3.3 Blackout Periods

An alert that is received during a blackout period is suppressed. That is, it is received by Alerta and a `202 Accepted` status code is returned however this means that even though the alert has been accepted, it won't be processed.

Alerta defines many different alert attributes that can be used to group alerts and it is these attributes that can be used to define blackout rules. For example, to suppress alerts from an entire environment, service or group, or a combination of these. However, it is possible to define blackout rules based only on resource and event attributes for situations that require that level of granularity.

Tags can also be used to define a blackout rule which should allow a lot of flexibility because tags can be added at source, using the alerta CLI, or using a plugin. Note that one or more tags can be required to match an alert for the suppression to apply.

In summary, blackout rules can be any of:

- an entire environment eg. `environment=Production`
- a particular resource eg. `resource=host55`
- an entire service eg. `service=Web`
- every occurrence of a specific event eg. `event=DiskFull`
- a group of events eg. `group=Syslog`
- a specific event for a resource eg. `resource=host55 and event=DiskFull`
- all events that have a specific set of tags eg. `tags=[blackout, london]`

Note that an `environment` is always required to be defined for a blackout rule.

1.3.4 De-Duplication

When an alert with the same `environment-resource-event` combination is received with the **same** severity, the alert is de-duplicated.

This means that information from the de-duplicated alert is used to update key attributes of the existing alert (like `duplicateCount`, `repeat` flag, `value`, `text` and `lastReceiveTime`) and the new alert is not shown.

Alerts are sorted in the Alerta web UI by `lastReceiveTime` by default so that the most recent alerts will be displayed at the top regardless of whether they were new alerts or de-duplicated alerts.

1.3.5 Simple Correlation

Alerta implements what we call “simple correlation” – as opposed to [complex correlation](#) which is [much more involved](#). Simple correlation, in combination with de-duplication, provides straight-forward and effective ways to reduce the burden of managing an alert console.

With Alerta, there are two ways alerts can be correlated, namely:

1. When an alert with the same `environment-resource-event` combination is received with a **different severity**, then the alert is correlated.
2. When a alert with the same `environment-resource` combination is received with an event in the `correlate` list of related events with **any** severity, then the alert is correlated.

In both cases, this means that information from the correlated alert is used to update key attributes of the existing alert (like `severity`, `event`, `value`, `text` and `lastReceiveTime`) and the new alert is not shown.

1.3.6 State-based Browser

Alerta is called state-based because it will **automatically** *change the alert status* based on the current and previous severity of alerts and subsequent user actions.

The Alerta API will:

- only show the most recent state of any alert
- change the status of an alert to `closed` if a `normal`, `ok` or `cleared` is received
- change the status of a `closed` alert to `open` if the event reoccurs
- change the status of an `acknowledged` alert to `open` if the new severity is higher than the current `severity`
- update the `severity` and other key attributes of an alert when a more recent alert is received (see [correlation](#) and [deduplication](#))
- update the `trendIndication` attribute based on `previousSeverity` and current `severity` with either `moreSevere`, `lessSevere` or `noChange`
- update the `history` log following a `severity` or `status` change (see [alert history](#))

All of these automatic actions combine to ensure that important alerts are given the priority they deserve.

Note: To take full advantage of the state-based browser it is recommended to implement the timeout of `expired` alerts using the [House Keeping](#) script.

1.3.7 Alert History

Whenever an alert status or severity changes, that change is recorded in the alert [history](#) log. This is to allow operations staff follow the lifecycle of a particular alert, if necessary.

The alert history is visible in the *Alert Details* page of any alert and also by using the `alerta` command-line tool `history` sub-command.

For example, it will show whether an alert status change happened as a result of operator (external) action or an automatic [correlation](#) (auto) action.

1.3.8 Heartbeats

An Alerta *heartbeat* is a periodic HTTP request sent to the Alerta API to indicate normal operation of the origin of the heartbeat.

They can be used to ensure components of the Alerta monitoring system are operating normally or sent from any other source. As well as an `origin` they include a `timeout` in seconds (after which they will be considered stale), and optional tags and attributes.

They are visible in the Alerta console (*Heartbeats* page) and via the `alerta` command-line tool using the `heartbeat` sub-command to send them, and the `heartbeats` sub-command to view them.

Alerts can be generated from stale or slow heartbeats using `alerta heartbeats --alert`. For more information about generating alerts from heartbeats see the *heartbeats command* reference.

1.4 Alerta Web UI

The Alerta web UI console takes full advantage of the *state-based Alerta API* to ensure that the most important events at any given time are brought to the attention of operators.

The following sections explain how to get the most out of the Alerta web UI.

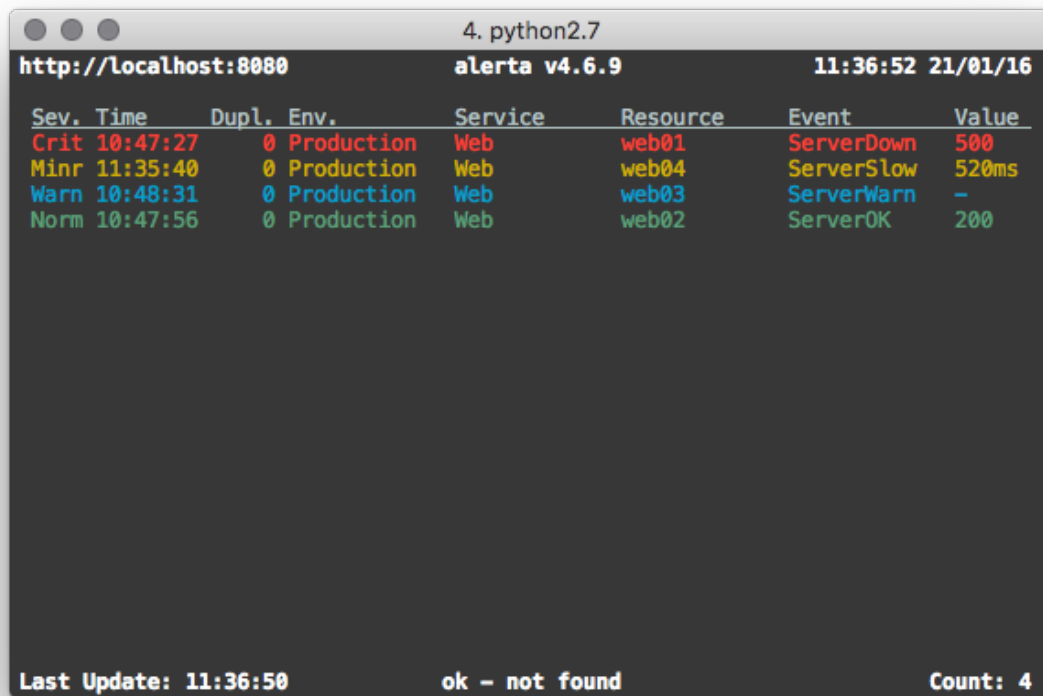
- Local and remote configuration
- Authentication methods
- Alert Summary List, Details & History
- Getting the most out of Heartbeats
- Managing Users & Groups
- Creating Blackout Periods
- Setting User Permissions
- Managing API Keys
- Understanding the Alert Reports
- User preference settings

Note: If you require help with any of the above information post a question on [Slack](#).

1.5 Alerta CLI

`alerta` is the unified command-line tool, terminal GUI and Python SDK for the alerta monitoring system.

It can be used to send and query alerts, tag alerts and change alert status, delete alerts, dump alert history or see the raw alert data. It can also be used to send heartbeats to the alerta server, and generate alerts based on missing or slow heartbeats.



The screenshot shows a terminal window titled '4. python2.7' with the URL 'http://localhost:8080' and 'alerta v4.6.9'. The timestamp is '11:36:52 21/01/16'. The table displays alert data with columns: Sev., Time, Dupl., Env., Service, Resource, Event, and Value. The data rows are: Crit 10:47:27, 0 Production, Web, web01, ServerDown, 500; Minr 11:35:40, 0 Production, Web, web04, ServerSlow, 520ms; Warn 10:48:31, 0 Production, Web, web03, ServerWarn, -; and Norm 10:47:56, 0 Production, Web, web02, ServerOK, 200. At the bottom, it shows 'Last Update: 11:36:50', 'ok - not found', and 'Count: 4'.

Sev.	Time	Dupl.	Env.	Service	Resource	Event	Value
Crit	10:47:27	0	Production	Web	web01	ServerDown	500
Minr	11:35:40	0	Production	Web	web04	ServerSlow	520ms
Warn	10:48:31	0	Production	Web	web03	ServerWarn	-
Norm	10:47:56	0	Production	Web	web02	ServerOK	200

Last Update: 11:36:50 ok - not found Count: 4

1.5.1 Installation

The alerta client tool can be installed using pip:

```
$ pip install alerta
```

Or, by cloning the git repository:

```
$ git clone https://github.com/alerta/python-alerta-client.git
$ cd python-alerta-client
$ pip install .
```

1.5.2 Configuration

Options can be set in a configuration file, as environment variables or on the command line. Profiles can be used to easily switch between different configuration settings.

Option	Config File	Environment Variable	Optional Argument	Default
file	n/a	ALERTA_CONF_FILE	--config-file FILE	~/.alerta.conf
profile	profile	ALERTA_DEFAULT_PROFILE	--profile PROFILE	None
endpoint	endpoint	ALERTA_ENDPOINT	--endpoint-url URL	http://localhost:8080
key	key	ALERTA_API_KEY	n/a	None
time-zone	timezone	n/a	n/a	Europe/London
timeout	timeout	n/a	n/a	5s TCP connection timeout
ssl verify	sslverify	REQUESTS_CA_BUNDLE	n/a	verify SSL certificates
output	output	n/a	--output FORMAT, --json	text
color	color	CLICOLOR	--color, --no-color	color on
debug	debug	DEBUG	--debug	no debug

Note: The profile option can only be set in the [DEFAULT] section.

Example

Configuration file ~/.alerta.conf:

```
[DEFAULT]
timezone = Australia/Sydney
output = json

[profile development]
endpoint = https://localhost:8443
key = demo-key
sslverify = off
timeout = 10.0
debug = yes
```

Set environment variables:

```
$ export ALERTA_CONF_FILE=~/.alerta.conf
$ export ALERTA_DEFAULT_PROFILE=production
```

Use production configuration settings by default:

```
$ alerta query
```

Switch to development configuration settings when required:

```
$ alerta --profile development query
```

1.5.3 Precedence

Command-line configuration options have precedence over environment variables, which have precedence over the configuration file. Within the configuration file, profile-specific sections have precedence over the [DEFAULT] section.

1.5.4 Authentication

If the Alerta API enforces authentication, then the `alerta` command-line tool can be configured to present an API key or login to the API before accessing secured endpoints.

API Keys

API keys can be generated in the web UI, or by an authenticated user using the `alerta` CLI, and should be added to the configuration file as the “key” setting as shown in the following example:

```
[profile production]
endpoint = https://api.alerta.io
key = LMvzLsfJyGpSuLmaB9kp-8gCl4I3YZkV4i7IGb6S
```

OAuth Login

Alternatively, a user can “login” to the API and retrieve a Bearer token if the Alerta API is configured to use either `basic`, `github`, `gitlab` or `google` as the authentication provider. No additional settings are required but before running any commands the user must login first:

```
$ alerta login
```

1.5.5 Commands

The `alerta` tool is invoked by specifying a command using the following format:

```
$ alerta [OPTIONS] COMMAND [ARGS]...
```

- *Alert Commands*
 - **send** - Send an alert
 - **query** - Search for alerts
 - **ack** - Acknowledge alerts
 - **close** - Close alerts
 - **unack** - Un-acknowledge alerts
 - **shelve** - Shelve alerts
 - **unshelve** - Un-shelve alerts
 - **tag** - Tag alerts
 - **untag** - Untag alerts
 - **update** - Update alert attributes
 - **delete** - Delete alerts
 - **watch** - Watch alerts

- **top** - Show top offenders and stats
- **raw** - Show alert raw data
- **history** - Show alert history
- *Blackout Commands*
 - **blackout** - Suppress alerts
 - **blackouts** - List alert suppressions
- *Heartbeat Commands*
 - **heartbeat** - Send a heartbeat
 - **heartbeats** - List heartbeats
- *API Key Commands*
 - **key** - Create API key
 - **keys** - List API keys
 - **revoke** - Revoke API key
- *User Commands*
 - **user** - Update user
 - **users** - List users
 - **me** - Update current user
- *Permissions Commands*
 - **perm** - Add role-permission lookup
 - **perms** - List role-permission lookups
- *Customer Commands*
 - **customer** - Add customer lookup
 - **customers** - List customer lookups
- *Auth Commands*
 - **signup** - Sign-up new user
 - **login** - Login with user credentials
 - **logout** - Clear login credentials
 - **whoami** - Display current logged in user
 - **token** - Display current auth token
- *Admin Commands*
 - **status** - Display status and metrics
 - **config** - Display remote client config
 - **housekeeping** - Expired and clears old alerts
 - **uptime** - Display server uptime
 - **version** - Display version info

- *Help Commands*
 - *help* - Show this help

Alert Commands

The following group of commands are related to sending, querying and managing the status of alerts.

send - Send an alert

Send an alert.

```
$ alerta send [OPTIONS]
```

Options:

-r, --resource RESOURCE	Resource under alarm
-e, --event EVENT	Event name
-E, --environment ENVIRONMENT	Environment eg. Production, Development
-s, --severity SEVERITY	Severity eg. critical, major, minor, warning
-C, --correlate EVENT	List of related events eg. node_up, node_down
-S, --service SERVICE	List of affected services eg. app name, Web, Network, Storage, Database, Security
-g, --group GROUP	Group event by type eg. OS, Performance
-v, --value VALUE	Event value
-t, --text DESCRIPTION	Description of alert
-T, --tag TAG	List of tags eg. London, os:linux, AWS/EC2
-A, --attributes KEY=VALUE	List of attributes eg. priority=high
-O, --origin ORIGIN	Origin of alert in form app/host
--type EVENT_TYPE	Event type eg. exceptionAlert, performanceAlert, nagiosAlert
--timeout SECONDS	Seconds before an open alert will be expired
--raw-data STRING	Raw data of original alert eg. SNMP trap PDU. '@' to read from file, '-' to read from stdin
--customer STRING	Customer
-h, --help	Show this message and exit.

The only mandatory options are resource and event. All the others will be set to sensible defaults.

Attention: If the reject plugin is enabled (which it is by default) then alerts must have an `environment` attribute that is one of either `Production` or `Development` and it must define a `service` attribute. For more information on configuring or disabling this plugin see [Plugin Settings](#).

Attribute	Default
environment	empty string
severity	normal
correlate	empty list
status	unknown
service	empty list
group	Misc
value	n/a
text	empty string
tags	empty list
attributes	empty dictionary
origin	program/host
type	exceptionAlert
timeout	86400 (1 day)
raw data	empty string

Examples

To send a minor alert followed by a normal alert that correlates:

```
$ alerta send --resource web01 --event HttpError --correlate HttpOK --group Web --
↪severity minor
$ alerta send --resource web01 --event HttpOK --correlate HttpError --group Web --
↪severity normal
```

To send an alert with custom attribute called region:

```
$ alerta send -r web01 -e HttpError -g Web -s major --attributes region="EU"
```

query - Search for alerts

Query for alerts based on search filter criteria.

Examples

To query for major and minor open alerts for the Production environment of the Mobile API service:

```
$ alerta query --filter severity=major --filter severity=minor --filter status=open --
↪filter environment=Production --filter service="Mobile API"
```

To query for all alerts with “disk” in the alert text:

```
$ alerta query --filter text=~disk
```

ack - Acknowledge alerts

Acknowledge alerts ie. change alert status to ack

close - Close alerts

Close alerts ie. change alert status to closed.

unack - Un-acknowledge alerts

Unacknowledge alerts ie. change alert status to open.

shelve - Shelve alerts

Shelve alerts ie. change alert status to shelved which removes the alerts from the active console and prevents any further notifications.

unshelve - Un-shelve alerts

Unshelve alerts ie. change alert status to open which returns the alerts to the active console and does not prevent future notifications.

tag - Tag alerts

Add tags to alerts.

untag - Untag alerts

Remove tags from alerts.

update - Update alert attributes

Update alert attributes.

delete - Delete alerts

Delete alerts.

watch - Watch alerts

Watch for new alerts.

top - Show top offenders and stats

Display alerts like unix “top” command.

raw - Show alert raw data

Show raw data for alerts.

history - Show alert history

Show action, status, severity and value changes for alerts.

Blackout Commands

The following group of commands are related to creating and managing alert suppressions using blackouts.

blackout - Suppress alerts

blackout Suppress alerts

blackouts - List alert suppressions

blackouts List alert suppressions

Heartbeat Commands

The following group of commands are related to creating and managing heartbeats.

heartbeat - Send a heartbeat

Send or delete a heartbeat.

```
$ alerta heartbeat [OPTIONS]
```

Options:

```
-O, --origin ORIGIN      Origin of heartbeat.  
-E, --environment ENVIRONMENT  Environment eg. Production, Development
```

(continues on next page)

(continued from previous page)

<code>-s, --severity SEVERITY</code>	Severity eg. critical, major, minor, warning
<code>-S, --service SERVICE</code>	List of affected services eg. app name, Web, Network, Storage, Database, Security
<code>-g, --group GROUP</code>	Group event by type eg. OS, Performance
<code>-T, --tag TAG</code>	List of tags eg. London, os:linux, AWS/EC2
<code>--timeout SECONDS</code>	Seconds before heartbeat is stale
<code>--customer STRING</code>	Customer
<code>-D, --delete ID</code>	Delete heartbeat using ID
<code>-h, --help</code>	Show this message and exit.

Note: The “environment”, “severity”, “service” and “group” values are only used when heartbeat alerts are generated from slow or stale heartbeats.

heartbeats - List heartbeats

List heartbeats and generate heartbeat alerts.

```
$ alerta heartbeats [OPTIONS]
```

Options:

<code>--alert</code>	Alert on stale or slow heartbeats
<code>-s, --severity SEVERITY</code>	Severity for stale heartbeat alerts
<code>--timeout SECONDS</code>	Seconds before a stale heartbeat alert will be expired
<code>--purge</code>	Delete all stale heartbeats
<code>-h, --help</code>	Show this message and exit.

Alerts can be generated from stale or slow heartbeats using the `--alert` option. It is expected that this would be run at regular intervals using a scheduling service such as `cron`.

Tags can be used to set the `environment` or `group` of a heartbeat alert using the format `environment:[ENV]` and `group:[GRP]`. These tags will be visible in the heartbeat but removed as tags from the alert.

Example

To send a major alert with an environment of `Infra` in the `Network` group when a heartbeat is missed or slow for an origin called `system1`:

```
$ alerta heartbeat -O system1 -T environment:Infra -T group:Network --timeout 10
(wait >10 seconds)
$ alerta heartbeats --alert --severity major
```

API Key Commands

The following group of commands are related to creating and managing API keys.

key - Create API key

key Create API key

Important: To prevent privilege escalation it is not possible to create an API key with associated roles that are greater than that with which that API key has.

keys - List API keys

keys List API keys

revoke - Revoke API key

revoke Revoke API key

User Commands

The following group of commands are related to creating and managing users.

user - Update user

user Update user

users - List users

users List users

me - Update current user

me Update current user

Permissions Commands

The following group of commands are related to creating and managing roles, permissions and access control.

perm - Add role-permission lookup

perm Add role-permission lookup

perms - List role-permission lookups

perms List role-permission lookups

Customer Commands

The following group of commands are related to creating and managing customers.

customer - Add customer lookup

customer Add customer lookup

customers - List customer lookups

customers List customer lookups

Auth Commands

The following group of commands are related to authentication.

signup - Sign-up new user

signup Sign-up new user

login - Login with user credentials

login Login with user credentials

logout - Clear login credentials

logout Clear login credentials

whoami - Display current logged in user

whoami Display current logged in user

token - Display current auth token

```
token Display current auth token
```

Admin Commands

The following group of commands are related to administration.

status - Display status and metrics

Display API server switch status and usage metrics.

\$ alerta status				
METRIC	TYPE	NAME	VALUE	AVERAGE

Total alerts	gauge	alerts.total	993	
Rejected alerts	counter	alerts.rejected	22	
Alert queries	timer	alerts.queries	9132459	128.713
Pre-receive plugins	timer	plugins.prereceive	10889	0.0383874
Newly created alerts	timer	alerts.create	4442	5.06123
Post-receive plugins	timer	plugins.postreceive	10867	0.0149995
Received alerts	timer	alerts.received	15376	23.4729
Duplicate alerts	timer	alerts.duplicate	9167	8.26061
Correlated alerts	timer	alerts.correlate	429	20.9068
Tagging alerts	timer	alerts.tagged	246	35.5935
Alert status change	timer	alerts.status	687	88.2969
Deleted alerts	timer	alerts.deleted	8	120.25
Removing tags from alerts	timer	alerts.untagged	52	22.2308
Count alerts	timer	alerts.counts	4388289	23.9553
Alerta console auto-refresh	text	switch.auto-refresh-allow	ON	
API alert submission	text	switch.sender-api-allow	ON	

config - Display remote client config

Display client config downloaded from API server.

```
$ alerta config
audio           : {}
auth_required   : True
client_id       : 736147134702-glkb1pesv716jlutg4llg7c3rr7nnhli.apps.
↳ googleusercontent.com
colors          : {}
customer_views  : True
dates           : {'longDate': 'EEEE, MMMM d, yyyy h:mm:ss.sss a (Z)', 'mediumDate':
↳ 'medium', 'shortTime': 'shortTime'}
endpoint        : https://alerta-api.herokuapp.com
github_url      : None
gitlab_url      : https://gitlab.com
keycloak_realm  : None
keycloak_url    : None
```

(continues on next page)

(continued from previous page)

```
pingfederate_url    : None
provider            : google
refresh_interval    : 5000
severity            : {'cleared': 5, 'critical': 1, 'debug': 7, 'indeterminate': 5,
↳ 'informational': 6, 'major': 2, 'minor': 3, 'normal': 5, 'ok': 5, 'security': 0, 'trace
↳ ': 8, 'unknown': 9, 'warning': 4}
signup_enabled      : True
tracking_id         : UA-44644195-5
```

housekeeping - Expired and clears old alerts

Trigger the expiration and deletion of alerts.

uptime - Display server uptime

Show how long the Alerta API has been running.

```
$ alerta uptime
01:06 up 0 days 16:15
```

version - Display version info

Show version information for alerta and dependencies.

```
$ alerta version
alerta 6.0.0
alerta client 6.0.0
requests 2.19.1
click 7.0
```

Help Commands

help - Show this help

Show all OPTIONS, COMMANDS and some example FILTERS.

1.5.6 Bugs

Log any issues on [GitHub](#) or submit a [pull request](#).

1.6 Integrations & Plugins

There are several different ways to integrate other alert sources into Alerta.

Firstly, existing *integrations* with well known monitoring tools like [Nagios](#), [Zabbix](#) and [Sensu](#) make use of the Alerta API and demonstrate how to build integrations with other monitoring tools.

Secondly, there are built-in *webhooks* for [AWS Cloudwatch](#), [Pingdom](#), [PagerDuty](#), [Google Stackdriver](#), [Prometheus Alertmanager](#) and more which provide ‘out-of-the-box’ integrations for some of the most popular monitoring systems available.

Thirdly, *alert severity indicators* or widgets can be placed on any web page using [oEmbed](#) for easy integration with existing dashboards.

Lastly, plugins can be used to quickly and easily forward alerts to or notify other systems like Slack or Hipchat.

Contents

- *Integrations*
 - *Core*
 - *Contrib*
 - *External*
 - *Bi-directional*
- *Webhooks*
 - *AWS CloudWatch*
 - *Grafana*
 - *Graylog*
 - *New Relic*
 - *PagerDuty*
 - *Pingdom*
 - *Prometheus Alertmanager*
 - *Riemann*
 - *SeverDensity*
 - *Slack*
 - *Google Stackdriver*
 - *Telegram*
- *Widgets*

1.6.1 Integrations

Core

There are a few core integrations which have been developed to showcase how easy it is to get alerts or events from other tools into Alerta. They are:

- [Nagios Event Broker](#) - forward host/service check results with suppression during downtime
- [InfluxData Kapacitor](#) - forward alerts for metric anomalies and dynamic thresholds
- [Zabbix Alert Script](#) - forward problems, acknowledged and OK events
- [Sensu Plugin](#) - forward Sensu events
- [Riemann Plugin](#) - generate alerts from thresholds defined against metric streams
- [Kibana Logging](#) - log alerts to Elasticsearch for historical visualisation of alert trends

Contrib

There are several more integrations available in the [contrib](#) repo which may be useful. They are:

- [Amazon SQS](#) - receive alerts from SQS that were sent using the SNS core plugin
- [E-mail](#) - send emails after a hold-time has expired (requires the ``AMQP`` message queue core plugin)
- [Opsweekly](#) - query Alerta to generate Opsweekly reports
- [Pinger](#) - generate ping alerts from list of network resources being pinged
- [SNMP Trap](#) - generate alerts from SNMPv1 and SNMPv2 sources
- [Supervisor](#) - trigger alerts and heartbeats based on process daemon events
- [Syslog Forwarder](#) - receive [RFC 5424](#), [RFC 3164](#) syslog and [Cisco](#) syslog messages
- [URL monitor](#) - trigger alerts from web service query responses

External

Some third-party monitoring tools have built-in support for Alerta. They are:

- [elastalert](#) - alerting on anomalies, spikes, or other patterns of interest from data in Elasticsearch
- [netdata](#) - a system for distributed real-time performance and health monitoring
- [Tick Stack](#) - designed to handle metrics and events using Telegraf, InfluxDB, Chronograf, and Kapacitor

Bi-directional

Bi-directional integration is where the system integrating with Alerta provides information that enables Alerta to link back to the originating system, either via an external link or simply a hostname and reference ID.

There are several examples of this two-way integration and they mostly take advantage of the flexible nature of the the `tags` and `attributes` alert attributes which can be used to keep track of the external reference.

Usage

In it's simplest form, pass the URL of the external system that generated the alert in an attribute called `externalUrl` (or similar):

```
$ alerta send -E ... --attribute externalUrl=https://my.example.com/go?id=1234
```

Better still, surround the URL with HTML markup to make the link clickable in the web UI:

```
$ alerta send -E ... --attribute externalUrl='<a href="https://my.example.com/go?id=1234
↪">ref 1234</a>'
```

Examples

The following is a list of integrations, webhooks and plugins that highlight the use of bi-directional integration in different ways.

- AWS Cloudwatch webhook - includes the [SNS subscription confirmation](#) link in the text of the alert
- Zabbix integration & plugin - TBC
- Grafana webhook - includes [rule and image links](#) in Grafana alert attributes if available
- NewRelic webhook - includes [incident and runbook links](#) in NewRelic alerts
- PagerDuty webhook - includes the [incident URL](#) in alert history text when status changes
- Prometheus webhook - includes [external and generator URLs](#) in the alert attributes
- Zabbix integration - includes [moreInfo](#) link back to Zabbix console event trigger page in alert attribute

1.6.2 Webhooks

Webhooks are a way of integrating with other systems by triggering [HTTP callbacks](#) to the Alerta server API when an event occurs.

Built-in Webhooks

- *AWS CloudWatch*
- *Grafana*
- *Graylog*
- *New Relic*
- *PagerDuty*
- *Pingdom*
- *Prometheus Alertmanager*
- *Riemann*
- *SeverDensity*
- *Slack*
- *Google Stackdriver*
- *Telegram*

Note: If authentication is enforced, then an API key is needed to access the alerta API programmatically and use the webhooks.

Please follow this page for more information on how to pass your api-key : <https://docs.alerta.io/en/latest/authentication.html#api-keys>

AWS CloudWatch

Alerta can be configured to receive AWS CloudWatch alarms by subscribing the Alerta API endpoint to an SNS topic.

For details on how to set this up see the [Sending Amazon SNS Messages to HTTP/HTTPS Endpoints](#) page and in the *Endpoint* input box append `/webhooks/cloudwatch` to the Alerta API URL.

Example AWS CloudWatch Webhook URL

`https://alerta.example.com/api/webhooks/cloudwatch`

Example AWS CloudWatch Webhook URL with authentication

`https://alerta.example.com/api/webhooks/cloudwatch?api-key=xxxxx`

Grafana

Alerta can be configured to receive Grafana alerts by adding a webhook endpoint to the Notification Channels.

For details on how to set this up see [Grafana webhook](#) page and in the *Endpoint URL* input box append `/webhooks/grafana` to the Alerta API URL.

Example Grafana Webhook URL

`https://alerta.example.com/api/webhooks/grafana`

The following parameters can be set in the url environment, event_type, group, origin, service, severity, timeout

Example Grafana Webhook URL with parameters

`https://alerta.example.com/api/webhooks/grafana?api-key=xxx&environment=Production&service=Web&timeout=`

Graylog

TBC

New Relic

Alerta can be configured to receive New Relic incidents by adding a webhook endpoint to the Notification Channels.

For details on how to set this up see [New Relic webhook](#) page and in the *Endpoint URL* input box append `/webhooks/newrelic` to the Alerta API URL.

Example New Relic Webhook URL

`https://alerta.example.com/api/webhooks/newrelic`

Example New Relic Webhook URL with authentication

`https://alerta.example.com/api/webhooks/newrelic?api-key=xxxxx`

PagerDuty

Alerta can be configured to receive PagerDuty incident-based webhooks – any change to the status or `assigned_to_user` of an incident will cause an outgoing message to be sent.

For details on how to set this up see the [PagerDuty webhook](#) page and where it requires the webhook URL append `/webhooks/pagerduty` to the Alerta API URL.

Example PagerDuty Webhook URL

```
https://alerta.example.com/api/webhooks/pagerduty
```

Example PagerDuty Webhook URL with authentication

```
https://alerta.example.com/api/webhooks/pagerduty?api-key=xxxxx
```

Pingdom

Alerta can be configured to receive Pingdom URL check alerts by adding a webhook alerting endpoint that calls the Alerta API.

For details on how to set this up see the [Pingdom webhook](#) page and in the *webhook URL* input box append `/webhooks/pingdom` to the Alerta API URL.

Example Pingdom Webhook URL

```
https://alerta.example.com/api/webhooks/pingdom
```

Example Pingdom Webhook URL with authentication

```
https://alerta.example.com/api/webhooks/pingdom?api-key=xxxx
```

Prometheus Alertmanager

Alerta can be configured as a webhook receiver in Alertmanager.

For details on how to set this up see the [Prometheus Config GitHub Repo](#)

Riemann

Alerta can be configured to receive Riemann events. The integration makes no assumptions about the format of the Riemann events and consumes standard events. If events are decorated with additional metadata (eg. tags, environment, group, etc) then these will be used.

Example Riemann Webhook URL

```
https://alerta.example.com/api/webhooks/riemann
```

Example Riemann Webhook URL with authentication

```
https://alerta.example.com/api/webhooks/riemann?api-key=xxxxx
```

SeverDensity

Alerta can be configured to receive SeverDensity alerts by adding a webhook endpoint to the Notification Preferences. For details on how to set this up see [SeverDensity webhook](#) page and in the *Endpoint URL* input box append / webhooks/serverdensity to the Alerta API URL.

Example SeverDensity Webhook URL

`https://alerta.example.com/api/webhooks/serverdensity`

Example SeverDensity Webhook URL with authentication

`https://alerta.example.com/api/webhooks/serverdensity?api-key=xxxxx`

Slack

TBC

Google Stackdriver

Alerta can be configured to receive Google Stackdriver incidents by adding a webhook endpoint to the notifications configuration.

For details on how to set this up see [Stackdriver webhook](#) page and in the *ENDPOINT URL* input box append / webhooks/stackdriver to the Alerta API URL.

Example Stackdriver Webhook URL

`https://alerta.example.com/api/webhooks/stackdriver`

Example Stackdriver Webhook URL with authentication

`https://alerta.example.com/api/webhooks/stackdriver?api-key=xxxxx`

Telegram

Alerta can be configured to receive [Telegram callback queries](#) from the inline buttons in the ``Telegram Bot`_` plugin.

For details on how to set this up see ``Telegram Bot`_` page and for the TELEGRAM_WEBHOOK_URL setting append / webhooks/telegram to the Alerta API URL.

Example Telegram Webhook URL

`https://alerta.example.com/api/webhooks/telegram`

Example Telegram Webhook URL with authentication

`https://alerta.example.com/api/webhooks/telegram?api-key=xxxxx`

1.6.3 Widgets

Add an alert severity indicator (aka. widget) to any dashboard using the Oembed API endpoint. The severity indicator is coloured with the maximum severity for that alert query filter and has a count for the total number of matching alerts for each severity.

Multiple severity indicators can be placed on the same page each for a different environment, service or group. See the [example oembed web page](#).

1.7 Configuration

The following settings are configured on the Alerta server. For `alerta` CLI configuration options see the [command-line reference](#) and for Web UI configuration options see the [web UI reference](#).

The configuration file uses standard python syntax for setting variables. The default settings (defined in `settings.py`) **should not** be modified directly.

To change any of these settings create a configuration file that overrides these default settings. The default location for the server configuration file is `/etc/alertad.conf` however the location itself can be overridden by using a environment variable `ALERTA_SVR_CONF_FILE`.

For example, to set the blackout period default duration to 1 day (ie. 86400 seconds):

```
$ export ALERTA_SVR_CONF_FILE=~/.alertad.conf
$ echo "BLACKOUT_DURATION = 86400" >$ALERTA_SVR_CONF_FILE
```

1.7.1 Config File Settings

- *General Settings*
- *Logging Settings*
- *API Settings*
- *Search Settings*
- *Database Settings*
- *Bulk API Settings*
- *Authentication Settings*
- *Auth Provider Settings*
- *Basic Auth Settings*
- *LDAP Auth Settings*
- *OpenID Connect Auth Settings*
- *SAML 2.0 Auth Settings*
- *Azure Active Directory Auth Settings*
- *Amazon Cognito Auth Settings*
- *GitHub Auth Settings*
- *GitLab Auth Settings*

- *Google Auth Settings*
- *Keycloak Auth Settings*
- *API Key & Bearer Token Settings*
- *HMAC Auth Settings*
- *Audit Log Settings*
- *CORS Settings*
- *Severity Settings*
- *Timeout Settings*
- *Housekeeping Settings*
- *Email Settings*
- *Web UI Settings*
- *Alert Status Indicator Settings*
- *Plugin Settings*

General Settings

Example

```
DEBUG = True
SECRET_KEY = 'changeme'
BASE_URL = '/api'
USE_PROXYFIX = False
```

DEBUG debug mode for increased logging (default is False)

SECRET_KEY a unique, randomly generated sequence of ASCII characters.

BASE_URL if API served on a path or behind a proxy use it to fix relative links (no default)

USE_PROXYFIX if API served behind SSL terminating proxy (default is False)

Logging Settings

Example

```
LOG_HANDLERS = ['console', 'file']
LOG_FILE = '/var/log/alertad.log'
LOG_MAX_BYTES = 5*1024*1024 # 5 MB
LOG_BACKUP_COUNT = 2
LOG_FORMAT = 'verbose'
```

LOG_CONFIG_FILE full path to logging configuration file in `dictConfig` format (default logging config)

LOG_HANDLERS list of log handlers eg. 'console', 'file', 'wsgi' (default is 'console')

LOG_FILE full path to write rotating server log file (default is `alertad.log`)

LOG_LEVEL only log messages with log severity level or higher (default is `WARNING`)

LOG_MAX_BYTES maximum size of log file before rollover (default is 10 MB)

LOG_BACKUP_COUNT number of rollover files before older files are deleted (default is 2)

LOG_FORMAT log file formatter name eg. 'default', 'simple', 'verbose', 'json'

LOG_METHODS only log listed HTTP methods eg. 'GET', 'POST', 'PUT', 'DELETE' (default is all HTTP methods)

API Settings

Example

```
ALARM_MODEL='ALERTA'
DEFAULT_PAGE_SIZE = 1000
HISTORY_LIMIT = 100
HISTORY_ON_VALUE_CHANGE = False # do not log if only value changes
```

ALARM_MODEL set to ISA_18_2 to use experimental ANSI/ISA 18.2 alarm model (default is ALERTA)

DEFAULT_PAGE_SIZE maximum number of alerts returned in a single query (default is 1000)

HISTORY_LIMIT number of history entries for each alert before old entries are deleted (default is 100)

HISTORY_ON_VALUE_CHANGE create history entry for duplicate alerts if value changes (default is True)

Search Settings

Example

```
DEFAULT_FIELD = 'text'
```

DEFAULT_FIELD search default field when no field given when using *query string syntax* (default is text)

Database Settings

There is a choice of either Postgres or MongoDB as the backend database.

Note: Development first began using MongoDB and then Postgres support was added later. At present, new features are tested against Postgres first and then ported to MongoDB. Both backends have extensive tests to ensure they are functionally equivalent however there are still minor differences in how each implements some search features.

The database is defined using the standard database connection URL formats. Many database configuration options are supported as connection URL parameters.

Postgres Example

```
DATABASE_URL = 'postgresql://other@localhost/otherdb?connect_timeout=10&application_
↪name=myapp'
DATABASE_NAME = 'monitoring'
```

See [Postgres connection strings](#) for more information.

MongoDB Example

```
DATABASE_URL = 'mongodb://db1.example.net,db2.example.net:2500/?replicaSet=test&
↳connectTimeoutMS=300000'
DATABASE_NAME = 'monitoring'
DATABASE_RAISE_ON_ERROR = False # creating tables & indexes manually
```

See [MongoDB connection strings](#) for more information.

DATABASE_URL database connection string (default is `mongodb://localhost:27017/monitoring`)

DATABASE_NAME database name can be used to override database in connection string (no default)

DATABASE_RAISE_ON_ERROR terminate startup if database configuration fails (default is `True`)

Bulk API Settings

The bulk API requires a Celery backend and can be used to off-load long-running tasks. (experimental)

Example Redis Task Queue

```
BULK_QUERY_LIMIT = 10000
CELERY_BROKER_URL='redis://localhost:6379/0'
CELERY_RESULT_BACKEND='redis://localhost:6379/0'
```

BULK_QUERY_LIMIT limit the number of tasks in a single bulk query (default is `100000`)

CELERY_BROKER_URL URL of Celery-supported broker (no default)

CELERY_RESULT_BACKEND URL of Celery-supported result backend (no default)

Authentication Settings

If enabled, authentication provides additional benefits beyond just security, such as auditing, and features like the ability to assign and watch alerts.

Example

```
AUTH_REQUIRED = True
ADMIN_USERS = ['admin@alerta.io', 'devops@example.com']
DEFAULT_ADMIN_ROLE = 'ops'
ADMIN_ROLES = ['ops', 'devops', 'coolkids']
USER_DEFAULT_SCOPES = ['read', 'write:alerts']
CUSTOMER_VIEWS = True
```

AUTH_REQUIRED users must authenticate when using web UI or command-line tool (default `False`)

ADMIN_USERS email addresses or logins that are assigned the “admin” role

DEFAULT_ADMIN_ROLE default role name used by **ADMIN_ROLES** (default is `admin`)

ADMIN_ROLES list of “roles” or “groups” that are assigned the “admin” role (default is a list containing the `DEFAULT_ADMIN_ROLE`)

USER_DEFAULT_SCOPES default permissions assigned to logged in users (default is `['read', 'write']`)

GUEST_DEFAULT_SCOPES default permissions assigned to guest users (default is `['read:alerts']`)

CUSTOMER_VIEWS enable [multi-tenancy](#) based on `customer` attribute (default is `False`)

Auth Provider Settings

Example

```
AUTH_PROVIDER = 'basic'
```

AUTH_PROVIDER valid authentication providers are basic, ldap, openid, saml2, azure, cognito, github, gitlab, google, keycloak, and pingfederate (default is basic)

Note: Any authentication provider that is [OpenID Connect compliant](#) is supported. Set the AUTH_PROVIDER to openid and configure the required OIDC settings [below](#).

Basic Auth Settings

Example

```
AUTH_PROVIDER = 'basic'
BASIC_AUTH_REALM = 'Monitoring'
SIGNUP_ENABLED = True
ALLOWED_EMAIL_DOMAINS = ['alerta.io', 'alerta.dev']
```

BASIC_AUTH_REALM BasicAuth authentication realm (default is Alerta)

SIGNUP_ENABLED prevent self-service sign-up of new users via the web UI (default is True)

ALLOWED_EMAIL_DOMAINS authorised email domains when using email as login (default is *)

LDAP Auth Settings

Example

```
AUTH_PROVIDER = 'ldap'
LDAP_URL = 'ldap://openldap:389'
LDAP_DOMAINS = {
    'my-domain.com': 'cn=%s,dc=my-domain,dc=com'
}
```

LDAP_URL URL of the LDAP server (no default)

LDAP_DOMAINS dictionary of LDAP domains and LDAP search filters (no default)

LDAP_DOMAINS_GROUP (default is empty dict {})

LDAP_DOMAINS_BASEDN (default is empty dict {})

LDAP_ALLOW_SELF_SIGNED_CERT (default is False)

OpenID Connect Auth Settings

OAuth2_CLIENT_ID client ID required by OAuth2 providers (no default)

OAuth2_CLIENT_SECRET client secret required by OAuth2 providers (no default)

OIDC_ISSUER_URL issuer URL also known as Discovery Document is used to auto-discover all necessary auth endpoints for an OIDC client (no default)

OIDC_LOGOUT_URL (no default)

OIDC_VERIFY_TOKEN (default is False)

OIDC_ROLE_CLAIM (default is roles)

OIDC_GROUP_CLAIM (default is groups)

ALLOWED_OIDC_ROLES (default is *)

ALLOWED_EMAIL_DOMAINS authorised email domains when using email as login (default is *)

SAML 2.0 Auth Settings

SAML2_ENTITY_ID (no default)

SAML2_METADATA_URL (no default)

SAML2_USER_NAME_FORMAT Python format string which will be rendered to user's name using SAML attributes. See saml2 (default is '{givenName} {surname}')

SAML2_EMAIL_ATTRIBUTE (default is 'emailAddress')

SAML2_CONFIG pysaml2 configuration dict. See saml2 (no default)

ALLOWED_SAML2_GROUPS list of authorised groups a user must belong to. See saml2 for details (default is *)

ALLOWED_EMAIL_DOMAINS authorised email domains when using email as login (default is *)

Azure Active Directory Auth Settings

Example

```
AZURE_TENANT = 'common'
OAUTH2_CLIENT_ID = 'd8de5642-52e5-480e-abab-9db88e9e341f'
OAUTH2_CLIENT_SECRET = 'a7Xx6eV~-4XUjycF.-9Lxw53N46G.L_ra0'
ALLOWED_EMAIL_DOMAINS = 'alerta.dev'
ADMIN_USERS = 'admin@alerta.dev'
```

AZURE_TENANT “common”, “organizations”, “consumers” or tenant ID (default is common)

Amazon Cognito Auth Settings

AWS_REGION AWS region (default is `us-east-1`)

COGNITO_USER_POOL_ID (no default)

COGNITO_DOMAIN (no default)

GitHub Auth Settings

GITHUB_URL API URL for public or privately run GitHub Enterprise server (default is `https://github.com`)

ALLOWED_GITHUB_ORGS authorised GitHub organisations a user must belong to (default is `*`)

GitLab Auth Settings

GITLAB_URL API URL for public or privately run GitLab server (default is `https://gitlab.com`)

ALLOWED_GITLAB_GROUPS authorised GitLab groups a user must belong to (default is `*`)

Google Auth Settings

OAUTH2_CLIENT_ID client ID required by OAuth2 providers (no default)

OAUTH2_CLIENT_SECRET client secret required by OAuth2 providers (no default)

ALLOWED_EMAIL_DOMAINS authorised email domains when using email as login (default is `*`)

Keycloak Auth Settings

KEYCLOAK_URL Keycloak website URL when using Keycloak as OAuth2 provider (no default)

KEYCLOAK_REALM Keycloak realm when using Keycloak as OAuth2 provider (no default)

ALLOWED_KEYCLOAK_ROLES list of authorised roles a user must belong to (no default)

API Key & Bearer Token Settings

TOKEN_EXPIRE_DAYS number of days a web UI bearer token is valid (default is 14)

API_KEY_EXPIRE_DAYS number of days an API key is valid (default is 365)

HMAC Auth Settings

Example

```

HMAC_AUTH_CREDENTIALS = [
    # {
    #     'id': "", # access key id => $ uuidgen | tr '[:upper:]' '[:lower:]'
    #     'key': "", # secret key => $ date | md5 | base64
    #     'algorithm': 'sha256' # valid hmac algorithm eg. sha256, sha384, sha512
    # }
] # type: List[Dict[str, Any]]

```

HMAC_AUTH_CREDENTIALS HMAC credentials

Audit Log Settings

Audit events can be logged locally to the standard application log (which could also help with general debugging) or forwarded to a HTTP endpoint using a POST.

Example

```
AUDIT_TRAIL = ['admin', 'write', 'auth']
AUDIT_LOG = True # log to Flask application logger
AUDIT_LOG_REDACT = True
AUDIT_LOG_JSON = False
AUDIT_URL = 'https://listener.logz.io:8071/?token=TOKEN'
```

AUDIT_TRAIL audit trail for admin, write or auth changes. (default is ['admin'])

AUDIT_LOG enable audit logging to configured application log file (default is False)

AUDIT_LOG_REDACT redact sensitive data before logging (default is True)

AUDIT_LOG_JSON log alert data as JSON object (default is False)

AUDIT_URL forward audit logs to HTTP POST URL (no default)

CORS Settings

Example

```
CORS_ORIGINS = [
    'http://localhost',
    'http://localhost:8000',
    r'https?://\w*\.?local\.alerta\.io:?\d*/?.*' # => http(s)://*.local.alerta.io:<port>
]
```

CORS_ORIGINS URL origins that can access the API for Cross-Origin Resource Sharing (CORS)

Severity Settings

The severities and their order are customisable to fit with the environment in which Alerta is deployed.

Example

```
SEVERITY_MAP = {
    'critical': 1,
    'warning': 4,
    'indeterminate': 5,
    'ok': 5,
    'unknown': 9
}
DEFAULT_NORMAL_SEVERITY = 'ok' # 'normal', 'ok', 'cleared'
DEFAULT_PREVIOUS_SEVERITY = 'indeterminate'

COLOR_MAP = {
    'severity': {
        'critical': 'red',
        'warning': '#1E90FF',
```

(continues on next page)

(continued from previous page)

```

        'indeterminate': 'lightblue',
        'ok': '#00CC00',
        'unknown': 'silver'
    },
    'text': 'black',
    'highlight': 'skyblue '
}

```

SEVERITY_MAP dictionary of severity names and levels

DEFAULT_NORMAL_SEVERITY severity to be assigned to new alerts (default is normal)

DEFAULT_PREVIOUS_SEVERITY previous severity to be assigned to new alerts (default is indeterminate)

COLOR_MAP dictionary of severity colors, text and highlight color

Timeout Settings

Alert timeouts are important for housekeeping and heartbeat timeouts are important for generating alerts from stale heartbeats.

Example

```

ALERT_TIMEOUT = 43200 # 12 hours
HEARTBEAT_TIMEOUT = 7200 # 2 hours
HEARTBEAT_MAX_LATENCY

```

ALERT_TIMEOUT default timeout period in seconds for alerts (default is 86400)

HEARTBEAT_TIMEOUT default timeout period in seconds for heartbeats (default is 86400)

HEARTBEAT_MAX_LATENCY stale heartbeat threshold in milliseconds (default is 2000)

Housekeeping Settings

Example

```

DEFAULT_EXPIRED_DELETE_HRS = 12 # hours
DEFAULT_INFO_DELETE_HRS = 0 # do not delete info alerts

```

DEFAULT_EXPIRED_DELETE_HRS delete expired alerts after defined hours (0=do not delete, default is 2)

DEFAULT_INFO_DELETE_HRS delete informational alerts after defined hours (0=do not delete, default is 12)

Email Settings

If email verification is enabled then emails are sent to users when they sign up via BasicAuth. They must click on the provided link to verify their email address before they can login.

Example

```

EMAIL_VERIFICATION = True
SMTP_HOST = 'smtp.example.com'
MAIL_FROM = 'noreply@alerta.io'

```

EMAIL_VERIFICATION enforce email verification of new users (default is False)

SMTP_HOST SMTP host of mail server (default is smtp.gmail.com)

SMTP_PORT SMTP port of mail server (default is 587)

MAIL_LOCALHOST mail server to use in HELO/EHLO command (default is localhost)

SMTP_STARTTLS SMTP connection in TLS (Transport Layer Security) mode. All SMTP commands that follow will be encrypted (default is False)

SMTP_USE_SSL used for situations where SSL is required from the beginning of the connection and using SMTP_STARTTLS is not appropriate (default is False)

SSL_KEY_FILE a PEM formatted private key file for the SSL connection(no default)

SSL_CERT_FILE a PEM formatted certificate chain file for the SSL connection (no default)

MAIL_FROM valid email address from which emails are sent (no default)

SMTP_USERNAME application-specific username, if different to MAIL_FROM user (no default)

SMTP_PASSWORD application-specific password for MAIL_FROM or SMTP_USERNAME (no default)

Web UI Settings

The following settings are specific to the web UI and are not used by the server.

Example

```
SITE_LOGO_URL = 'http://pigment.github.io/fake-logos/logos/vector/color/fast-banana.svg'
DATE_FORMAT_SHORT_TIME = 'HH:mm'
DATE_FORMAT_MEDIUM_DATE = 'EEE d MMM HH:mm'
DATE_FORMAT_LONG_DATE = 'd/M/yyyy h:mm:ss.sss a'
DEFAULT_AUDIO_FILE = '/audio/Bike Horn.mp3'
COLUMNS = ['severity', 'status', 'lastReceiveTime', 'duplicateCount',
            'customer', 'environment', 'service', 'resource', 'event', 'value', 'text']
SORT_LIST_BY = 'lastReceiveTime'
ACTIONS = ['createIssue', 'updateIssue']
GOOGLE_TRACKING_ID = 'UA-44644195-5'
AUTO_REFRESH_INTERVAL = 30000 # 30s
```

SITE_LOGO_URL URL of company logo to replace “alerta” in navigation bar (no default)

DATE_FORMAT_SHORT_TIME format used for time in columns eg. 09:24 (default is HH:mm)

DATE_FORMAT_MEDIUM_DATE format used for dates in columns eg. Tue 9 Oct 09:24 (default is EEE d MMM HH:mm)

DATE_FORMAT_LONG_DATE format used for date and time in detail views eg. 9/10/2018 9:24:03.036 AM (default is d/M/yyyy h:mm:ss.sss a)

DEFAULT_AUDIO_FILE make sound when new alert arrives. must exist on client at relative path eg. /audio/Bike Horn.mp3 (no default)

COLUMNS user defined columns and column order for alert list view (default is standard web console column order)

SORT_LIST_BY to sort by newest use lastReceiveTime or oldest use -createTime. minus means reverse (default is lastReceiveTime)

DEFAULT_FILTER default alert list filter as query filter (default is {'status':['open','ack']})

ACTIONS adds buttons to web console for operators to trigger custom actions against alert (no default)

GOOGLE_TRACKING_ID used by the web UI to send tracking data to Google Analytics (no default)

AUTO_REFRESH_INTERVAL interval in milliseconds at which the web UI refreshes alert list (default is 5000)

Alert Status Indicator Settings

Example

```
ASI_SEVERITY = [
    'critical', 'major', 'minor', 'warning', 'indeterminate', 'informational'
]
ASI_QUERIES = [
    {'text': 'Production', 'query': [['environment', 'Production']]},
    {'text': 'Development', 'query': [['environment', 'Development']]},
    {'text': 'Heartbeats', 'query': {'q': 'event:Heartbeat'}},
    {'text': 'Misc.', 'query': 'group=Misc'},
]
```

ASI_SEVERITY severity counts to include in status indicator (default is all non-normal)

ASI_QUERIES list of alert queries applied to filter status indicators (see example for default)

Plugin Settings

Plugins are used to extend the behaviour of the Alerta server without having to modify the core application. The only plugins that are installed and enabled by default are the `reject` and `blackout` plugins. Other plugins are available in the [contrib repo](#).

Example

```
PLUGINS = ['reject', 'blackout', 'slack']
PLUGINS_RAISE_ON_ERROR = False # keep processing other plugins if exception
```

PLUGINS list of enabled plugins (default ['reject', 'blackout'])

PLUGINS_RAISE_ON_ERROR stop processing plugins if there is an exception (default is True)

Reject Plugin Settings

Alerts can be rejected based on the `origin` or `environment` alert attributes.

Example

```
ORIGIN_BLACKLIST = ['foo/bar$', '.*/*ux'] # reject all foo alerts from bar, and
↳ everything from qux
ALLOWED_ENVIRONMENTS = ['Production', 'Development', 'Testing']
```

ORIGIN_BLACKLIST list of alert origins blacklisted from submitting alerts. useful for rouge alert sources (no default)

ALLOWED_ENVIRONMENTS list of allowed environments. useful for enforcing discrete set of environments (default is ['Production', 'Development'])

Note: To disable the `reject` plugin simply remove it from the list of enabled plugins in the `PLUGINS` configuration setting to override the default.

Blackout Plugin Settings

Alerts can be suppressed based on alert attributes for arbitrary durations known as “blackout periods”. An alert received during a blackout period is rejected, by default.

Example

```
BLACKOUT_DURATION = 7200 # 2 hours
NOTIFICATION_BLACKOUT = True
BLACKOUT_ACCEPT = ['normal', 'ok', 'cleared']
```

BLACKOUT_DURATION default period for an alert blackout (default is 3600)

NOTIFICATION_BLACKOUT instead of rejecting alerts received during blackout periods, set status of alert to blackout and do not forward to plugins (default is False)

BLACKOUT_ACCEPT used with NOTIFICATION_BLACKOUT if alerts with status of blackout should still be closed by “ok” alerts (no default)

1.7.2 Environment Variables

Some configuration settings are special because they can be overridden by environment variables. This is to make deployment to different platforms and managed environments such as Heroku, Kubernetes and AWS easier, or to make use of managed Postgres or MongoDB services.

Note: Environment variables are read after configuration files so they will always override any other setting.

General Settings

DEBUG see above

BASE_URL see above

USE_PROXYFIX see above

SECRET_KEY see above

AUTH_REQUIRED see above

AUTH_PROVIDER see above

ADMIN_USERS see above

CUSTOMER_VIEWS see above

OAuth2_CLIENT_ID see above

OAuth2_CLIENT_SECRET see above

ALLOWED_EMAIL_DOMAINS see above

GITHUB_URL see above

ALLOWED_GITHUB_ORGS see above

GITLAB_URL see above

ALLOWED_GITLAB_GROUPS see above

KEYCLOAK_URL see above

KEYCLOAK_REALM see above

ALLOWED_KEYCLOAK_ROLES *see above*

PINGFEDERATE_OPENID_ACCESS_TOKEN_URL *see above*

PINGFEDERATE_OPENID_PAYLOAD_USERNAME *see above*

PINGFEDERATE_OPENID_PAYLOAD_EMAIL *see above*

PINGFEDERATE_OPENID_PAYLOAD_GROUP *see above*

PINGFEDERATE_PUBKEY_LOCATION *see above*

PINGFEDERATE_TOKEN_ALGORITHM *see above*

CORS_ORIGINS *see above*

MAIL_FROM *see above*

SMTP_PASSWORD *see above*

GOOGLE_TRACKING_ID *see above*

PLUGINS *see above*

Database Settings

DATABASE_URL used by both Postgres and MongoDB for database connection strings

DATABASE_NAME database name can be used to override default database defined in DATABASE_URL

MongoDB Settings

Deprecated since version 5.0: Use DATABASE_URL and DATABASE_NAME instead.

MONGO_URI used to override MONGO_URI config variable using the standard connection string format

MONGODB_URI alternative name for MONGO_URI environment variable which is used by some managed services

MONGOHQ_URL automatically set when using [Heroku MongoHQ](#) managed service

MONGOLAB_URI automatically set when using [Heroku MongoLab](#) managed service

MONGO_PORT automatically set when deploying [Alerta to a Docker](#) linked mongo container

1.7.3 Dynamic Settings

Using the *management switchboard* on the API some dynamic settings can be switched on and off without restarting the Alerta server daemon.

Currently, there is only one setting that can be toggled in this way and it is the Auto-refresh allow switch.

Auto-Refresh Allow

The Alerta Web UI will automatically refresh the list of alerts in the alert console every 5 seconds.

If for whatever reason, the Alerta API is experiencing heavy load the `auto_refresh_allow` switch can be turned off and the Web UI will respect that and switch to manual refresh mode. The Alerta web UI will start auto-refreshing again if the `auto_refresh_allow` switch is turned back on.

1.8 Authentication

By default, authentication is not enabled, however there are some features that are not available unless users login such as watching alerts.

Alerta supports five main authentication strategies:

- *Basic Auth*
- *LDAP*
- *OpenID Connect*
- *SAML 2.0*
- *OAuth2 (Note: only used by GitHub)*

The OpenID Connect authentication strategy can be used to integrate with any [OIDC compliant](#) auth provider however Alerta has specific implementations for some auth providers to simplify the integration:

- *Azure Active Directory*
- *Amazon Cognito*
- *GitLab OAuth2*
- *Google OAuth2*
- *Keycloak OAuth2*

Alerta also supports two “machine-to-machine” authentication strategies:

- *API Keys*
- *HMAC Auth*

To enforce authentication set `AUTH_REQUIRED` to `True` and set the `SECRET_KEY` to some random string in the `alertad.conf` server configuration settings file:

```
AUTH_REQUIRED = True
SECRET_KEY = 'ZWU2YTU0Zjg2MDkyY2RmYmRlNDM4MjYzNWQzMWMxYzQK'
```

Note: Ensure that the `SECRET_KEY` that is used to encode tokens and API keys is a unique, randomly generated sequence of ASCII characters. The following command generates a suitable 32-character random string on Linux:

```
$ LC_CTYPE=C tr -dc A-Za-z0-9_!\@#\$\%^&*()\-+= < /dev/urandom | head -c 32 && echo
```

or Mac OSX:

```
$ date | md5 | base64
```

1.8.1 Basic Auth

Basic Auth (built-in)

The most straight-forward authentication strategy to implement of the four is HTTP [Basic Authentication](#) provided by the Alerta API because there is no additional configuration required of the Alerta server to use it other than setting `AUTH_REQUIRED` to `True`.

Note: HTTP Basic Auth does not provide any encryption of the username or password so it is strongly advised to only use Basic Auth over HTTPS.

Warning: add example

Basic Auth using LDAP

LDAP can be used as the Basic Auth provider to authenticate users if desired by setting the `AUTH_PROVIDER` to `ldap`. It requires the installation of an additional Python package called “`python-ldap`” and can be installed using:

```
$ pip install python-ldap
```

Important: If the Alerta API is installed in a Python virtual environment ensure that the `python-ldap` package is installed into the same environment otherwise it won't be auto-detected.

The configuration settings for LDAP authentication include the LDAP server URL and a map of LDAP domains to search filters which means that multiple LDAP domains can be supported.

Example

```
AUTH_PROVIDER = 'ldap'
LDAP_URL = 'ldap://localhost:389' # replace with your LDAP server
LDAP_DOMAINS = {
    'my-domain.com': 'uid=%s,ou=users,dc=my-domain,dc=com'
}
LDAP_DOMAINS_BASEDN = {
    'my-domain.com': 'dc=my-domain,dc=com'
}
LDAP_DOMAINS_GROUP = {
    'my-domain.com': '(&(memberUid={username})(objectClass=groupOfUniqueNames))'
    #OR
    'my-domain.com': '(&(member={userdn})(objectClass=groupOfUniqueNames))'
    #OR
    'my-domain.com': '(&(member={email})(objectClass=groupOfUniqueNames))'
}
```

Warning: improve example

A typical user called `user1`, for the example above, would login using an email address of `user1@my-domain.com` even if that email address doesn't actually exist.

You can fetch ldap groups dynamically from LDAP server and use them as customer name by using `LDAP_DOMAINS_GROUP` configuration. Either of `{username}`, `{userdn}` or `{email}` can be used for the same.

All users are initially assigned the "user" role by default.

Note: User sign-up, email verification and password reset through the Alerta web UI or CLI is not supported. Self-service user management needs to be handled by the LDAP authentication provider.

1.8.2 OpenID Connect

```
# OpenID Connect
OIDC_ISSUER_URL = None
OIDC_AUTH_URL = None
OIDC_LOGOUT_URL = None
OIDC_VERIFY_TOKEN = False
OIDC_ROLE_CLAIM = OIDC_CUSTOM_CLAIM = 'roles' # JWT claim name whose value is used in
↪role mapping
OIDC_GROUP_CLAIM = 'groups' # JWT claim name whose value is used in customer mapping
ALLOWED_OIDC_ROLES = ALLOWED_GITLAB_GROUPS or ALLOWED_KEYCLOAK_ROLES or ['*']
```

1.8.3 SAML 2.0

To use SAML as the authentication provider for Alerta, install `PySAML2` on the Alerta server and follow the configuration steps below.

```
$ pip install PySAML2
```

Generate private/public key pair:

```
$ openssl req -utf8 -new -x509 -days 3652 -nodes -out "alerta.cert" -keyout "alerta.key"
```

Note: This key pair is not related to HTTPS.

Configure pysaml2:

Bare-minimum config example:

```
AUTH_PROVIDER = 'saml2'
SAML2_CONFIG = {
    'metadata': {
        'local': ['/path/to/federationmetadata.xml']
    },
    'key_file': '/path/to/alerta.key',
    'cert_file': '/path/to/alerta.cert'
}
```

metadata IdP metadata (refer to [saml2 documentation](#) for possible ways of specifying it)

key_file, cert_file path to aforementioned keys

Refer to pysaml2 documentation and source code if you need additional options:

- <https://pysaml2.readthedocs.io/en/latest/howto/config.html>
- <https://github.com/rohe/pysaml2/blob/master/src/saml2/config.py>

Note: entityid and service provider endpoints are configured by default based on your BASE_URL value which is mandatory if you use SAML (see general config)

ALLOWED_SAML2_GROUPS

To restrict access to users who are members of particular group use:

```
ALLOWED_SAML2_GROUPS = ['alerta_ro', 'alerta_rw']
```

Note: Ensure that pysaml2 authn response identity object contains groups attribute. You can do this by writing proper attribute map which will convert your IdP-specific attribute name to groups.

Example:

```
MAP = {
    ...
    'fro': {
        ...
        'http://schemas.xmlsoap.org/claims/group': 'groups',
        ...
    },
    'to': {
        ...
        'groups': 'http://schemas.xmlsoap.org/claims/group',
        ...
    }
}
```

See [pysaml2 attribute-map-dir documentation](#). The attribute-map-dir can be specified in the SAML2_CONFIG.

SAML2_USER_NAME_FORMAT

The username format can be customized using the SAML2_USER_NAME_FORMAT setting. It is a python string template which is used to generate user's name based on attributes (make sure that [attribute-map-dir](#) is properly configured in case default does not fit).

Default is '{givenName} {surname}'.

CORS_ORIGINS

You also need to add your IdP origin to CORS headers:

```
CORS_ORIGINS = [
    ...
    'https://sso.example.com',
    ...
]
```

Add trusted Service Provider to your Identity Provider

Your metadata url is: {BASE_URL}/auth/saml/metadata.xml, pass it to your IdP administrator.

```
# SAML 2.0
SAML2_ENTITY_ID = None
SAML2_METADATA_URL = None
SAML2_USER_NAME_FORMAT = '{givenName} {surname}'
SAML2_EMAIL_ATTRIBUTE = 'emailAddress'
SAML2_CONFIG = {} # type: Dict[str, Any]
ALLOWED_SAML2_GROUPS = ['*']
```

GitHub OAuth2

To use GitHub as the OAuth2 provider for Alerta, login to GitHub and go to *Settings -> Applications -> Register New Application*.

- Application Name: Alerta
- Homepage URL: <http://alerta.io>
- Application description (optional): Guardian Alerta monitoring system
- Authorization callback URL: <http://alerta.example.com>

Note: The *Authorization callback URL* is the most important setting and it is nothing more than the URL domain (ie. without any path) where the alerta Web UI is being hosted.

Click Register Application and take note of the Client ID and Client Secret. Then configuration settings for alerta server are as follows:

```
AUTH_PROVIDER = 'github'
OAUTH2_CLIENT_ID = 'f7b0c15e2b722e0e38f4'
OAUTH2_CLIENT_SECRET = '7aa9094369b72937910badab0424dc7393x8mpl3'
```

To restrict access to users who are members of particular GitHub organisations use:

```
ALLOWED_GITHUB_ORGS = ['example', 'mycompany']
```

Note: ALLOWED_GITHUB_ORGS can be an asterisk (*) to force login but *not* restrict who can login.

Important: To revoke access of your instance of alerta to your GitHub user info at any time go to *Settings -> Applications -> Authorized* applications, find alerta in the list of applications and click the **Revoke** button.

1.8.4 OIDC Providers

OpenID Connect authentication is provided by [Google `OAuth2`](#), [GitLab OAuth 2.0](#) or [Keycloak OAuth 2.0](#) and configuration is more involved than the Basic Auth setup.

Note: If Alerta is deployed to a publicly accessible web server it is important to configure the OAuth2 settings correctly to ensure that only authorised users can access and modify your alerts.

Ensure `AUTH_REQUIRED` and `SECRET_KEY` are set and that the `AUTH_PROVIDER` setting is set to the correct provider.

Then follow the steps below for the chosen OAuth provider to create an OAuth client ID and client secret. The client ID and client secret will need to be added to the `alertad.conf` file for the Alerta server.

Azure Active Directory

To use [Azure Active Directory](#) (now known as [Microsoft identity platform \(v2.0\)](#)) as the OpenID Connect authentication provider for Alerta follow the steps below.

1. **Login to Azure portal** <https://portal.azure.com/>
2. Navigate to “Azure Active Directory” service page
3. From the “Manage” sidebar choose “App registrations”
4. **Click the button that says “New registration”** Fill in the “Register an application” form for your environment:
 - Name: Alerta AD
 - Supported Account Types: Multitenant and Personal (common)
 - Redirect URI: (web) <https://alerta.example.com>

... and click the “Register” button.

Note: The `AZURE_TENANT` setting will vary depending on what “Supported Account Type” is chosen. It will be either “common”, “organizations”, “consumers” or a tenant ID. To check which account type click the “Endpoints” button on the “Overview” page and check the “OpenID Connect metadata document” URL.

Example of OpenID Connect metadata URL for “organizations”

<https://login.microsoftonline.com/organizations/v2.0/.well-known/openid-configuration>

Copy the App registration details for client ID, for example:

Application (client) ID: 3aab3fa8-cb9b-457f-8283-811d1ebd4975

5. From the “Manage” sidebar again choose “Certificates & secrets”

Click the “New client secret” button

Add description “Alerta Web UI” and choose an expiry time

Copy the client secret, for example:

jj2cw7~nc1.5513.UAy8C309Ng-.~GYWYp

6. Add the above details to the Alerta server configuration file, like so:

```
AZURE_TENANT = 'common'  
OAUTH2_CLIENT_ID = '3aab3fa8-cb9b-457f-8283-811d1ebd4975'  
OAUTH2_CLIENT_SECRET = 'jj2cw7~nc1.5513.UAy8C309Ng-.~GYWYp'
```

Amazon Cognito

Note: TBC

GitLab OAuth2

To use GitLab as the OAuth2 provider for Alerta, login to GitLab and go to *Profile Settings -> Applications -> New Application*.

- Name: Alerta
- Callback URL: <http://alerta.example.com>
- Scopes: openid

GitLab Projects Groups More

User Settings > Applications

Edit application

Name

Alerta

Redirect URI

http://local.alerta.io:8000
http://127.0.0.1:9004

Use one line per URI

Scopes

- ☐ **api**
Grants complete read/write access to the API, including all groups and projects.
- ☐ **read_user**
Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.
- ☐ **sudo**
Grants permission to perform API actions as any user in the system, when authenticated as an admin user.
- ☐ **read_repository**
Grants read-only access to repositories on private projects using Git-over-HTTP (not using the API).
- ☐ **read_registry**
Grants read-only access to container registry images on private projects.
- ☒ **openid**
Grants permission to authenticate with GitLab using OpenID Connect. Also gives read-only access to the user's profile and group memberships.

Save application

Note: The *Callback URL* is the most important setting and it is nothing more than the URL domain (ie. without any

path) where the alerta Web UI is being hosted.

Click *Submit* and take note of the Application ID and Secret. Then configuration settings for alerta server are as follows (replacing the values shown below with the values generated by GitLab):

```
AUTH_PROVIDER = 'gitlab'
GITLAB_URL = 'https://gitlab.com' # or your own GitLab server
OAUTH2_CLIENT_ID = 'd31e9caa131f72901b16d22289c824f423bd5cbf187a11245f402e8b2707d591'
OAUTH2_CLIENT_SECRET = '42f1de369ec706996cadda234986779eeb65c0201a6f286b9751b1f845d62c8a'
```

To restrict access to users who are members of particular GitLab groups use:

```
ALLOWED_GITLAB_GROUPS = ['group1', 'group2']
```

Note: ALLOWED_GITLAB_GROUPS can be an asterisk (*) to force login but *not* restrict who can login.

Important: To revoke access of your instance of alerta to your GitLab user info at any time go to *Profile Settings* -> *Applications* -> *Authorized applications*, find alerta in the list of applications and click the **Revoke** button.

Google OAuth2

To use Google as the OAuth2 provider for Alerta, login to the [Google Developer Console](#) and create a new project for alerta.

- Project Name: alerta
- Project ID: (automatically assigned)

Next go to *APIs & Services* and select *Credentials* from the sidebar menu. Click **Create credentials** and choose “OAuth client ID” and “Web Application” for application type.

- Name: Alerta
- Authorized Javascript Origins: <http://alerta.example.com>
- Authorized Redirect URIs: <http://alerta.example.com>

Click **Create** and take note of the Client ID and Client Secret. Use this information to configure the settings for alerta server.

Example

```
AUTH_PROVIDER = 'google'
OAUTH2_CLIENT_ID = '379647311730-sjl30ru952o3o7ig8u0ts8np2ojivr8d.apps.googleusercontent.
↪com'
OAUTH2_CLIENT_SECRET = '8HrqJhbrYn9oDtaJqExample'
```

or using ‘openid’:

```
AUTH_PROVIDER = 'openid'
OIDC_ISSUER_URL = 'https://accounts.google.com'
OAUTH2_CLIENT_ID = '379647311730-sjl30ru952o3o7ig8u0ts8np2ojivr8d.apps.googleusercontent.
↪com'
OAUTH2_CLIENT_SECRET = '8HrqJhbrYn9oDtaJqExample'
```

Deprecated since version 6.6: Google+ API is no longer a requirement.

Warning: It is no longer necessary to enable [Google+ API](#) to use Google OAuth. Google+ API will be shutdown on March 7, 2019 and Alerta installations configured to use Google+ API will cease to function after that date.

To restrict access to users with particular [Google apps domains](#) use:

```
ALLOWED_EMAIL_DOMAINS = ['example.org', 'mycompany.com']
```

Note: ALLOWED_EMAIL_DOMAINS can be an asterisk (*) to force login but *not* restrict who can login.

Keycloak OAuth2

To use Keycloak as the OAuth2 provider for Alerta, login to Keycloak admin interface, select the realm and go to *Clients* -> *Create*.

- Client ID: alerta-ui
- Client protocol: openid-connect
- Root URL: <http://alerta.example.org>

After the client is created, edit it and change the following properties:

- Access Type: confidential

Add the following mapper under the *Mappers* tab:

```
Name: role memberships
Mapper type: User Realm Role
Multivalued: ON
Token Claim Name: roles
Claim JSON type: String
Add to userinfo: ON
```

Now go to *Installation* and generate it by selecting 'Keycloak OIDC JSON'. You should get something like this:

```
{
  "realm": "master",
  "auth-server-url": "https://keycloak.example.org/auth",
  "ssl-required": "external",
  "resource": "alerta-ui",
  "credentials": {
    "secret": "418bbf31-aef-33d1-a471-322a60276879"
  },
  "use-resource-role-mappings": true
}
```

Take note of the realm, resource and secret. Then configuration settings for alerta server are as follows (replacing the values shown below with the values generated by Keycloak):

```
AUTH_PROVIDER = 'keycloak'
KEYCLOAK_URL = 'https://keycloak.example.org'
KEYCLOAK_REALM = 'master'
OAUTH2_CLIENT_ID = 'alerta-ui'
OAUTH2_CLIENT_SECRET = '418bbf31-aef-33d1-a471-322a60276879'
```

To restrict access to users who are associated with a particular [Keycloak role](#) use:

```
ALLOWED_KEYCLOAK_ROLES = ['role1', 'role2']
```

Note: ALLOWED_KEYCLOAK_ROLES can be an asterisk (*) to force login but *not* restrict who can login.

Note: When using self-hosted authentication providers, such as Keycloak, it may be necessary to set the REQUESTS_CA_BUNDLE environment variable, supported by the Python requests package, to the self-issued CA bundle to avoid [SSL verification issues](#).

1.8.5 API Keys

If authentication is enforced, then an API key is needed to access the alerta API programmatically. An API key can also be used by the [alerta CLI](#) for when the CLI is used in scripts. See the [example CLI config](#) for how to set the API key for the command-line tool.

Keys can be easily generated from the Alerta web UI and can have any scopes associated with them. They are valid for 1 year by default but this period is configurable using API_KEY_EXPIRE_DAYS in the server configuration.

To use an API key in an API query you must put the key in either an HTTP header or a query parameter.

Important: Using an HTTP header is the preferred method so that API keys are not exposed even when using HTTPS or inadvertently captured in log files.

Example using HTTP header

Use either the Authorization header with authorization type of Key:

```
$ curl 'http://api.alerta.io/alerts' -H 'Authorization: Key demo-key' -H 'Accept: application/json'
```

or the custom header X-API-Key:

```
$ curl 'http://api.alerta.io/alerts' -H 'X-API-Key: demo-key' -H 'Accept: application/json'
```

Example using query paramter

Use the api-key URL parameter:

```
$ curl 'http://api.alerta.io/alerts?api-key=demo-key' -H 'Accept: application/json'
```


1.8.6 HMAC Auth

Note: TBC

1.9 Authorization

Authorization is used to limit access to Alerta API resources. The authorization model is based on [Role-Based Access Control](#) (RBAC) which assigns permissions to functional roles and then users are assigned to one or more of those roles.

This “role-based access” allows for fine-grained control over exactly what resources are accessible to which users and exactly what type of access is allowed – in a way that is scalable.

For example, to create a new alert the sender will need to be assigned to at least one role with `write:alerts` permissions. If the sender is not a member of a role with those permissions then the request will be rejected with a 403 Forbidden response code.

1.9.1 Flat RBAC

Alerta implements the “flat” RBAC model as defined by NIST (aka. “Core RBAC”) and as such does not support any form of role hierarchy where roles can inherit other roles.

Flat RBAC has the following features:

- users acquire permissions through roles
- must support many-to-many user-role assignment
- must support many-to-many permission-role assignment
- must support user-role assignment review
- users can use permissions of multiple roles simultaneously

Note: All access is through roles. Permissions can not be assigned directly to users. The only exception to this is the `ADMIN_USERS` setting which overrides all other roles a user might belong to.

1.9.2 Configuration

There are two ways to configure role-based access; default and custom configuration.

Default Authorization

If *authentication* is enabled then the default authorization is used which defines two roles:

- **user** role - everyone is a “user” unless listed in the `ADMIN_USERS` setting. (default scopes are `read` and `write`)
- **admin** role - only admins can delete alerts and heartbeats, create users etc. (default scope is simply `admin`, however that implicitly includes `read` and `write`)

Note: The `user` and `admin` roles are protected preventing them from being deleted and preventing new roles from being created with the same names. The scopes associated with the default `user` role are managed using the `USER_DEFAULT_SCOPES` setting in the API *server settings*. All other roles are managed via the web console or `alerta` CLI.

Custom Authorization

To use custom authorization simply define one or more permission-scope lookups.

As an “admin” user go to *Configuration -> Permissions* and add a new role with the required scopes (see below for list of valid scopes). Those roles should match the roles, groups or organisations associated with users for the configured Authentication provider.

Tip: It is encouraged to employ the principle of least privilege when creating roles. That is, do not give to a user any more privilege than is necessary to perform their job function.

1.9.3 Scopes and Permissions

Use these scopes to request access to API resources.

Scope	Permissions
read	Grants read-only access to all scopes.
write	Grants read/write access to all scopes.
admin	Grants admin, read, write and delete access to all scopes.
read:alerts	Read-only access to alerts.
write:alerts	Grants read/write access to alerts.
delete:alerts (*)	Required to delete alerts, unless have admin access.
admin:alerts	Grants read/write to alerts for any customer.
read:blackouts	Grants read-only access to blackouts.
write:blackouts	Grants read/write access to blackouts.
admin:blackouts	Grants read/write access to blackouts for any customer.
read:heartbeats	Read-only access to heartbeats.
write:heartbeats	Grants read/write access to heartbeats.
admin:heartbeats	Grants read/write access to heartbeats for any customer.
write:users	Grant write access to users.
admin:users	Fully manage users.
read:perms	Grants read-only access to permissions and scopes.
admin:perms	Grants read, write and delete access to permissions.
read:customers	Grants read-only access to customers.
admin:customers	Fully manage customers.
read:keys	List and view API keys.
write:keys	Create, list and view API keys.
admin:keys	Fully manage API keys for any customer.
write:webhooks	Grants write access to webhooks.
read:oembed	Grants read-only to oembed endpoints.
read:management	Grants read-only access to management endpoints.
admin:management	Fully manage management endpoints.
read:userinfo	Grants read-only access to userinfo.

Note: write implicitly includes read, and admin implicitly includes read and write.

delete:alerts only required to delete alerts if the *DELETE_SCOPES* setting is enabled.

1.9.4 Audit Log

An audit trail can be enabled to keep track of changes to Alerta.

Every audit event will have an audit id, @timestamp, event, category, message, user, resource, request and extra elements. The extra element may include relevant data depending on the type of event.

Example Audit Event

```
{
  "id": "c87210da-3cfb-4cbd-b8ec-4fe9ed39aeef",
  "@timestamp": "2018-11-10T21:36:23.946Z",
  "event": "apikey-deleted",
  "category": "admin",
  "message": "",
  "user": {
    "id": "satterly",
```

(continues on next page)

(continued from previous page)

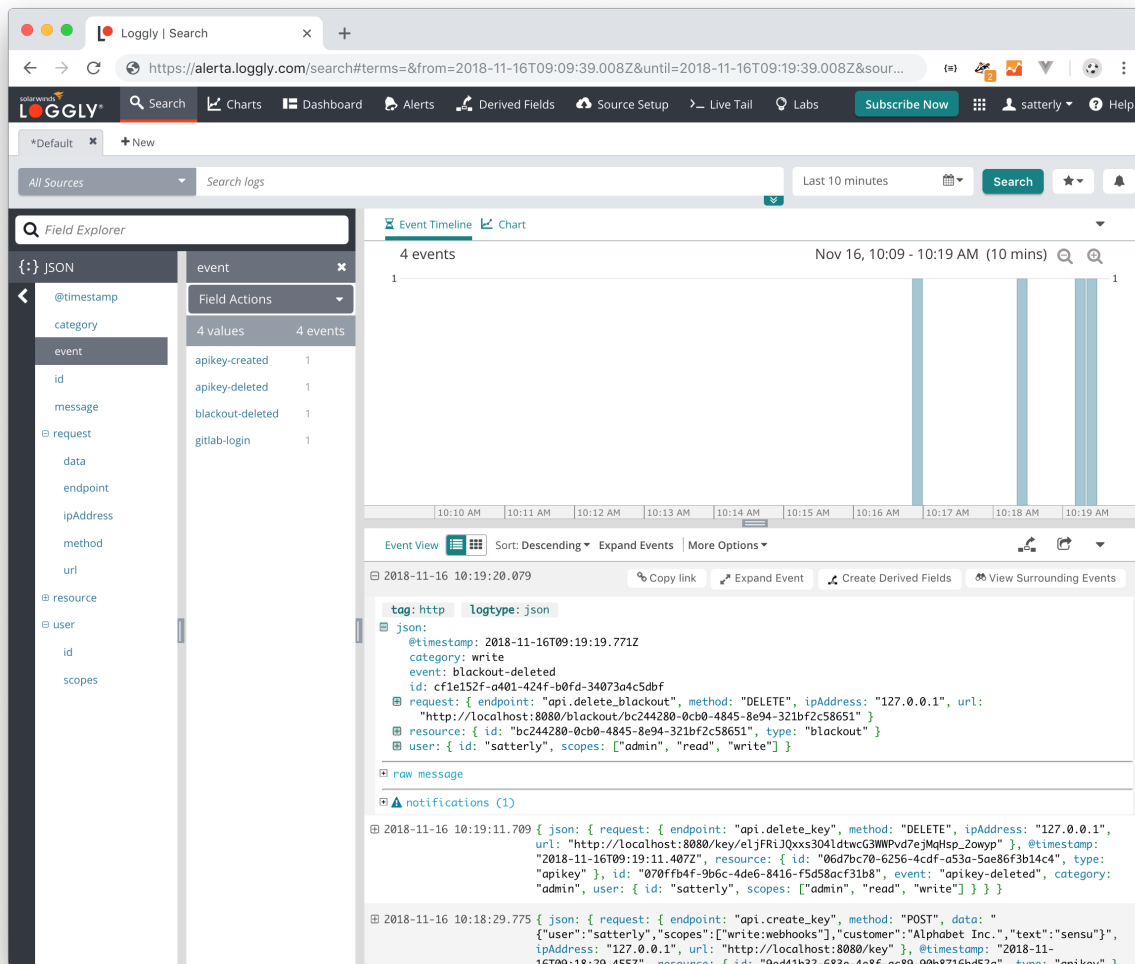
```
"customers": [],
"scopes": [
  "admin",
  "read",
  "write"
],
},
"resource": {
  "id": "dc0b5a62-015b-4ba3-965e-012ca2e4db9b",
  "type": "apikey"
},
"request": {
  "endpoint": "api.delete_key",
  "method": "DELETE",
  "url": "http://localhost:8080/key/dc0b5a62-015b-4ba3-965e-012ca2e4db9b",
  "args": {},
  "data": "",
  "ipAddress": "127.0.0.1"
},
"extra": {}
}
```

Audit events can be logged locally to the standard application log (which could also help with general debugging) or forwarded to a HTTP endpoint using a POST.

Example Loggly configuration

The following example configuration can be used to log all `admin`, `write` and `auth` requests to the Flask application log file and forward the events to the Loggly “logging-as-a-service” endpoint, replacing `TOKEN` in the Loggly URL with your customer token.

```
AUDIT_TRAIL = ['admin', 'write', 'auth']
AUDIT_LOG = True # log to Flask application logger
AUDIT_URL='http://logs-01.loggly.com/inputs/TOKEN/tag/http/'
```



1.10 Deployment

1.10.1 WSGI Server

There are many ways to deploy Alerta. It can be run as `alertad` during development or testing but when run in a production environment, it should *always be deployed* as a WSGI application. See the list of *real world* examples below for different ways to run Alerta as a WSGI application.

Note: When deploying with Apache `mod_wsgi`, be aware that by default Apache strips the Authentication header. This will cause you to receive “Missing authorization API Key or Bearer Token” errors. This can be fixed by setting `WSGIPassAuthorization On` in the configuration file for the site.

1.10.2 Web Proxy

Running the Alerta API behind a web proxy can greatly simplify the Web UI setup which means you can completely [avoid](#) the potential for any cross-origin issues.

Also, if you run the API on an HTTPS/SSL endpoint then it can reduce the possibility of [mixed content](#) errors when a web application hosted on a HTTP endpoint tries to access resources on an HTTPS endpoint.

Example API configuration (extract)

This example nginx server is configured to serve the web UI from the root / path and reverse-proxy API requests to /api to the WSGI application running on port 8080:

```
server {  
  
    listen 80 default_server deferred;  
  
    access_log /dev/stdout main;  
  
    location /api/ {  
        proxy_pass http://backend/;  
        proxy_set_header Host $host:$server_port;  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
    }  
  
    location / {  
        root /app;  
    }  
}  
  
upstream backend {  
    server localhost:8080 fail_timeout=0;  
}
```

The server configuration file `alertad.conf` for this setup would need to set `BASE_URL`:

```
BASE_URL = '/api'
```

Additionally add the `USE_PROXYFIX` setting to fix relative links in HTTP responses if the web proxy is used for SSL termination:

```
USE_PROXYFIX = True # use if proxy is terminating HTTPS traffic
```

And the web UI configuration file `config.json` would need the `endpoint` setting to match that:

```
{"endpoint": "/api"}
```

1.10.3 Static Website

The Alerta web UI is just a directory of static assets that can be served from any location. An easy and cheap way to serve the web UI is from an [Amazon S3 bucket](#) as a static website.

Note: Serving the Alerta web UI from a static web hosting site **will not work** unless that domain is listed in the CORS_ORIGINS Alerta API server configuration settings.

1.10.4 Authentication & SSL

Alerta supports several authentication mechanisms for both the API and the web UI and some key features of the web UI, like watching alerts, are only available if authentication is enabled.

The API can be secured using [API Keys](#) and the web UI can be secured using Basic Auth or an OAuth provider from either GitHub, GitLab, Google, Keycloak or SAML2.

If you plan to make the web UI accessible from a public URL it is strongly advised to [enforce authentication](#) and use HTTPS/SSL connections to the Alerta API to protect private alert data.

1.10.5 Authorisation & Customer Views

To restrict access to certain features use roles and [customer views](#).

1.10.6 Scalability

Alerta can scale horizontally, in the same way any other web application scales horizontally – a load balancer handles the HTTP requests and distributes those requests between all available application servers.

Note: If using multiple API servers ensure the same SECRET_KEY is used across all servers otherwise there will be problems with web UI user logins.

1.10.7 High Availability

To achieve high system availability the Alerta API should be deployed to scale out [horizontally](#) and the database should be deployed as a [replica set](#), if using mongoDB, or configure [replication](#), if using Postgres.

1.10.8 House Keeping

Deprecated since version 5.0: The `housekeepingAlerts.js` script that was used for housekeeping is deprecated. Use the following instead.

There are some jobs that should be run periodically to keep the Alerta console clutter free. To timeout *expired* alerts and delete old *closed* alerts you need to trigger housekeeping.

This can be done with the `alerta` command-line tool:

```
$ alerta housekeeping
```

This was not supported by earlier versions of the command-line tool and cURL has to be used to access `/management/housekeeping`.

The API key needs an admin scope if `AUTH_REQUIRED` is set to `True`.

It is suggested that you run housekeeping at regular intervals via `cron`. Every minute or two is a suitable interval.

By default, the housekeeping job will remove any alerts that have been expired or closed for 2 hours and any info messages that are 12 hours old. In some cases, these retention periods may be too long or too short for your needs.

Bear in mind that Alerta is intended to reflect the here and now, so long deletion thresholds should be avoided. Where you do need to depart from the defaults, you can specify like this:

```
$ alerta housekeeping --expired 2 --info 12
```

Heartbeats can be sent from any source to ensure that a system is 'alive'. To generate alerts for stale heartbeats the `alerta` command-line tool can be used:

```
$ alerta heartbeats --alert
```

Again, this should be run at regular intervals via `cron` or some other scheduler.

1.10.9 Management & Metrics

There are two management endpoints that provide internal application metrics.

The management endpoint `/management/status` can be used to keep track of realtime statistics on the performance of the Alerta API like alert counts and average processing time. For convenience, these statistics can be viewed in the *About* page of the Alerta web UI or using the `alerta` command-line tool `status` command.

The same metrics are also exposed at `/management/metrics` in the *exposition format* required by Prometheus so that it can be monitored by Prometheus and other monitoring tools that implement the *OpenMetrics* standard.

1.10.10 Web UI Analytics

Google analytics can be used to track usage of the Alerta web UI console. Just create a new tracking code with the *Google analytics* console and add it to the `alertad.conf` API configuration file:

```
GOOGLE_TRACKING_ID = 'UA-NNNNNN-N'
```

1.10.11 Real World Examples

Below are several different examples of how to run Alerta in production from a Debian *vagrant box*, an *AWS EC2 instance*, *Heroku PaaS* to a *Docker container*.

- *Vagrant* - deploy Alerta stand-alone or with Nagios, Zabbix, Riemann, Sensu or Kibana
- *Heroku* - deploy the Alerta API and the *web ui to Heroku PaaS*
- *AWS EC2* - deploy Alerta to EC2 using AWS Cloudformation
- *Docker* - deploy Alerta to a docker container
- *Docker Alpine* - full Alerta installation (including Mongo) based on Alpine Linux
- *Packer* - deploy Alerta to EC2 using Amazon AMIs
- *Flask deploy* - deploy Alerta as a generic Flask app

- [Ansible](#) - deploy Alerta using ansible on Centos 7
- [Terraform](#) - single instance of alerta for quick demo on AWS
- [Puppet](#) - Alerta recipe on top of [cfweb](#) module

1.11 Plug-ins

1.11.1 Plugins

[Plugin extensions](#) are an easy way of adding new features to Alerta that meet a specific end-user requirement.

Core

Core plugins have been developed as examples of common use-cases.

- [Reject](#) - reject alerts before processing. used to enforce custom alert format policies
- [Blackout](#) - suppression handler that will drop alerts that match a blackout period

Contrib

More than two dozen [contributed plugins](#) are made available for popular tools. Some of the most popular are:

- [AMQP](#) - publish alerts to an AMQP fanout topic after processing
- [Cachet](#) - create incidents for display on Cachet status page
- [Enhance](#) - add new information to an alert based on existing information
- [GeoIP Location](#) - use remote IP address to submitted alert to add location data
- [HipChat](#) - send alerts to HipChat room
- [InfluxDB](#) - send alerts to InfluxDB for graphing with Grafana
- [Logstash/Kibana](#) - send alerts to logstash agent after processing
- [Normalise](#) - ensure alerts are formatted in a consistent manner
- [PagerDuty Plugin](#) - send alerts to PagerDuty (webhooks used to receive callbacks)
- [Prometheus Silencer](#) - silence alerts in Prometheus Alertmanager if ack'ed in Alerta
- [Pushover.net](#) - send alerts to Pushover.net
- [Slack](#) - send alerts to Slack room
- [AWS SNS](#) - publish alerts to SNS topic after processing
- [Syslog Logger](#) - send alerts via syslog
- [Telegram Bot](#) - send alerts to Telegram channel
- [Twilio SMS](#) - send alerts via SMS using Twilio

custom error responses

1.12 Custom Webhooks

Custom webhooks are a simple but effective way of adding support for direct integration to any system via a webhook without having to modify the core source code. They are written in python and are required to implement all methods of a base class.

```
def incoming(self, path: str, query_string: ImmutableMultiDict, payload: Any) -> Union[Alert, JSON]:
    """
    Parse webhook path, query string and/or payload in JSON or plain text and
    return an alert or a custom JSON response.
    """
    raise NotImplementedError
```

They are loaded into memory when the Alerta API starts up and dynamically add an API endpoint path to the list of available webhooks at `/webhooks/<webhook_name>` and trigger for that and all subpaths of that URL. eg. `/webhooks/<webhook_name>/<alert_id>`

To set this up follow the instructions for triggering a webhook in the system to be integrated with and for the webhook URL append `/webhooks/<webhook_name>` to the Alerta API URL but replace `<webhook_name>` with the name of the system.

Next, write the webhook python code. For more information on how to write the python code see the [webhook examples](#) in the contrib repo or follow the tutorial.

Example Custom Webhook URL for Sentry

`https://alerta.example.com/api/webhooks/sentry`

Code for the webhook can be found in the contrib repo [Sentry webhook](#) directory.

1.13 Customer Views

Multitenancy is achieved using Customer views that are a way of ensuring logged in users only see alerts that relate to their organisation.

This is for Alerta deployments that are used to manage multiple customer sites.

1.13.1 Roles

The role of an API key is assumed to be “user” if it is a customer generated key. The role of admin

You can have a customer/user API key and an admin API key, but not a customer/admin API key – this makes no sense.

1.13.2 How it works

A new top-level alert attribute called `customer` is used to partition alerts for particular customers within the same alert database.

When a user logs in to the Alert console, a customer lookup is done to determine what customer value should be assigned to that user.

The customer value is then used as an implicit additional filter for all alert and heartbeat queries.

It is also assigned to any API keys generated by that user and the `customer` field is automatically used whenever that API key is used to generate or query for alerts.

1.13.3 Configuration

To configure customer views follow these three easy steps:

1. Authentication must be enforced and customer views enabled so in `alertad.conf`:

```
AUTH_REQUIRED = True
CUSTOMER_VIEWS = True
```

2. Define administrators that will have a global view of all customers and will have no restrictions on generating API keys or blackout periods, so in `alertad.conf`:

```
ADMIN_USERS = ['foo@bar.com']
```

3. Populate the Customer Lookup table in the web console to map Google email domains or Github/Gitlab orgs and groups to customers

1.13.4 Web Console for Users

Users that have a *customer view* are limited to what they can do in the web console (and via the API). They cannot create other users, configure blackout periods or modify the customer lookup table.

1.13.5 Web Console for Administrators

Administrators are not limited in what they can do in the web console (or via the API). Importantly, they can configure the customer lookup table.

1.14 Federated Alerta

Alerta servers can forward alerts, actions and deletes to other Alerta servers or downstream systems using the `forwarder` built-in plugin.

In this way, Alerta servers can be organised in a hierarchy of sub-ordinate servers forwarding to servers above them to create a hierarchy of Alerta servers.

Also individual Alerta servers can be kept in sync with others each other to create a “active-active” architectures where any server can be used as the “primary” and if a failure occurs any other server can take over.

Alerta can be configured to send some or all of the following:

- received alerts

- all alert actions or just specific actions. eg. ack, shelve, open, close
- custom alert actions eg. `createIssue` or `ringBell`
- alert deletions

Note: Forwarding heartbeats is not currently possible but may be supported in a future release.

1.14.1 Configuration

BASE_URL

Set the `BASE_URL` to the public endpoint that will be used by the web UI, alert generating systems, and forwarding Alerta servers

PLUGINS

Simply add the `fowarder` plugin to this list of configured plugins to enable and load it when the API starts.

FWD_DESTINATIONS

Forwarder destinations have three parts:

- remote `BASE_URL`
- authentication credentials
- forwarding filter

Example

```
FWD_DESTINATIONS = [  
    ('http://localhost:9000', {'username': 'user', 'password': 'pa55w0rd', 'timeout': 10}  
→, ['alerts', 'actions']), # BasicAuth  
    # ('https://httpbin.org/anything', dict(username='foo', password='bar', ssl_  
→verify=False), ['*']),  
    ('http://localhost:9001', {  
        'key': 'e3b8afc0-db18-4c51-865d-b95322742c5e',  
        'secret': 'MDhjZGMhYTRkY2YyNjk1MTEyMWFlNmM3Y2UxZDU1ZjIK'  
    }, ['actions']), # Hawk HMAC  
    ('http://localhost:9002', {'key': 'demo-key'}, ['delete']), # API key  
    ('http://localhost:9003', {'token': 'bearer-token'}, ['*']), # Bearer token  
]
```

1.14.2 Forwarding Loops

The configured remote `BASE_URL` must match the `BASE_URL` as set at the remote server otherwise forwarding loops are possible (see next).

There is no problem with Alerta servers forwarding alerts and actions to each other. This is the basis of an “active-active” failover configuration.

However, if misconfigured, servers can endlessly forward to each other and back again which would eventually lead to resource exhaustion and failure.

The solution is to borrow the concept of a “forwarding loop” header from mail servers. If the `X-Alerta-Loop` header contains the Alerta server name (ie `BASE_URL`) of the receiving server then the receiving server should not process the request.

And if an alert or action is to be forwarded to a remote Alerta server that is already in the “forwarding loop” header then that server should be skipped.

<https://qmail.mivzakim.net/qmail-manual-html/misc/RFCLOOPS.html>

1.14.3 Authentication

Alerta supports multiple authentication methods for the remote systems when forwarding:

- HMAC
- API Key
- BasicAuth
- Bearer token

The recommended authentication method for Alerta-to-Alerta forwarding is HMAC because it is the most secure, it does not require a user to be associated with the credentials and the credentials do not expire.

The other authentication methods are available for use when forwarding to non-Alerta systems.

To generate HMAC key and secret, it is useful to use a UUID for the key and base64 encoded string for the secret so that they are visibly different.

On macOS, run:

```
$ uuidgen | tr '[:upper:]' '[:lower:]'           <= create HMAC "key"
58e7c66f-b990-4610-9496-60eb3c63339b

$ date | md5 | base64                          <= create HMAC "secret"
MzVlMzQ5NWYzYWE2YTgxYTUyYmIyNDY0ZWE2ZWJlYTMK
```

1.14.4 Forwarding Filters

The types of entites to be forwarded are configurable:

- `*` - everything ie. alerts, all actions (incl. custom), deletes
- `alerts` - alerts
- `actions` - all actions
- individual actions - eg. `ack`
- custom actions eg. `createIssue`
- `delete`

Examples

```
['*']
['alerts', 'ack', 'unack', 'close', 'delete']
['alerts', 'delete']
```

1.14.5 Non-Alerta Forwarding

It is possible to make use of Alerta forwarding to forward alerts to non-Alerta systems. However, any forwarding destination will need to implement the following endpoints:

- POST /alert - alert receiver
- PUT /alert/{alertId}/action - action alerts
- DELETE /alert/{alertId} - delete alerts

Responses should have HTTP 200 OK status code on success. HTTP response bodies are not parsed so they will not effect the result. However, it is good practise to add meaningful messages to the payloads which will be useful when debugging.

1.14.6 Troubleshooting

- Enable detailed logging with DEBUG=True.
- Use a dummy endpoint such as <https://httpbin.org/anything>

1.14.7 Future Enhancements

- heartbeat forwarding
- circuit-breaker retry logic
- configurable endpoints

1.15 Conventions

Always favour convention over configuration. And any configuration should have sensible defaults.

1.15.1 Naming Conventions

Resources

The key alert attribute name of **resource** was specifically chosen so as not to be host centric. A resource *can* be a hostname, but it might also be an EC2 instance ID, a Docker container ID or some other type of non-host unique identifier.

Environments & Services

The environment attribute is used to **namespace** the alert resource. This allows you to have two resources with the same name (eg. `web01`) but that are differentiated by their environments (eg. `Production` and `Development`).

Choose a set of environments and enforce them. ie. `PROD`, `DEV` or `Production`, `Development` but not both. The same for services eg. `MobileAPI`, `Mobile-API` and `mobile api` are all valid but needlessly different and impossible to query for consistently or generate aggregate metrics for.

Note that the **service attribute is a list** because it is common for infrastructure (ie. some resource) to be used by more than one service. That is, if a component failure occurs that problem could cause an outage in multiple services.

Event Names

It can be useful to define a convention when it comes to naming events. Possible options are:

- Camel case - DiskUtilHigh
- Hierarchy - NW:INTERFACE:DOWN
- SNMP - cpuAlarmHigh

Querying for all Disk utilisation alerts using the alerta CLI is then relatively straight-forward:

```
$ alerta query --filter event=~DiskUtil
```

Event Groups

Another consideration is to ensure you make use of the event group which gives you the ability to group related alerts.

Some suggested event groups with possible events are listed below.

Event Groups	Events (examples)
Service	failures with entire services
Application	errors from application logs
OS	disk space, time sync failing
Performance	system load, swap utilisation high
Configuration	config mgmt tool alerts eg. Puppet or Chef
Web	web server errors
Syslog	unix system log messages
Hardware	hardware errors
Storage	NFS, SAN, NAS storage infrastructure
Database	database errors, table space utilisation
Security	security/authorization messages
Network	network devices and infrastructure
Cloud	cloud-based services or infrastructure

Querying for all performance-related alerts using the alerta CLI could then become:

```
$ alerta query --filter group=Performance
```

1.15.2 Severity Levels

Agree on a subset of *severity levels* and be consistent with what they mean. For example, if severity levels are used consistently then integrating with a paging or email system becomes easier.

Severity	Service Level	Notification
critical	service unavailable	immediate page out
major	service impaired still available	page during business hours
minor	component failure	email only
warning	everything else	consolidate into daily email

1.15.3 Enforcing Conventions

Once a set of naming conventions are agreed, they can be enforced by using a simple *pre-receive* plugin.

A full working example called `reject` can be found in the plugins directory of the project code repository and is installed by default. The server configuration settings `ORIGIN_BLACKLIST` and `ALLOWED_ENVIRONMENTS` can be used to tailor it for your circumstances or it can be disabled completely.

1.16 Development

1.16.1 Python SDK

Alerta is developed in Python so the Python SDK is a core component of the monitoring system.

Installation

Install using pip:

```
$ pip install alerta
```

Install master branch directly from GitHub:

```
$ pip install git+https://github.com/alerta/python-alerta-client.git@master
```

Examples

Initialise the alerta API client:

```
>>> from alertaclient.api import Client
>>> client = Client(endpoint='https://alerta-api.herokuapp.com/', key='demo-key')
```

Send an alert:

```
>>> client.send_alert(resource='foo', event='bar')
...
alertaclient.exceptions.UnknownError: [POLICY] Alert environment does not match one of
↳ Production, Development
```

Send an alert again, this time including the required environment and service:

```
>>> client.send_alert(resource='foo', event='bar', environment='Development', service=[
↳ 'Web'])
('fd3ecad4-6558-4ec7-96cc-aff6cdf1fab', Alert(id='fd3ecad4-6558-4ec7-96cc-aff6cdf1fab',
↳ environment='Development', resource='foo', event='bar', severity='normal', status=
↳ 'closed', customer=None), None)
```

Query for the alert just sent, by alert ID:

```
>>> client.get_alert('fd3ecad4-6558-4ec7-96cc-aff6cdf1fab')
Alert(id='fd3ecad4-6558-4ec7-96cc-aff6cdf1fab', environment='Development', resource='foo
↳ ', event='bar', severity='normal', status='closed', customer=None)
```


Search for alerts by attributes:

```
>>> client.get_alerts([('resource', 'foo'), ('environment', 'Development')])
[Alert(id='fd3ecad4-6558-4ec7-96cc-aff6cdf1fab', environment='Development', resource=
↳ 'foo', event='bar', severity='normal', status='closed', customer=None)]
```

Send a heartbeat:

```
>>> client.heartbeat(origin='baz')
Heartbeat(id='98c220e6-5148-4b19-8ae8-e1c078b7d68c', origin='baz', create_time=datetime.
↳ datetime(2018, 9, 6, 8, 48, 48, 817000), timeout=86400, customer=None)
```

For more details, visit the [Alerta Python SDK page](#).

1.16.2 Ruby SDK

The Ruby SDK is a work-in-progress. For more details, visit the [Alerta Ruby SDK page](#).

1.16.3 Haskell SDK

This SDK supplies bindings to the Alerta REST API so that it can be used from Haskell.

For more details, visit the [Haskell Package page](#).

1.16.4 Gource Visualization

View the development of Alerta over the years as an animated tree [Gource visualization](#).

1.17 Getting Started

The following tutorials are designed to get you started deploying and using Alerta in common scenarios.

1.17.1 Tutorials

- Deploy an Alerta Server
- Alert timeouts, heartbeats and housekeeping
- Use plugins to enhance Alerta
- Alerts explored in-depth
- Suppressing Alerts using Blackouts
- Authentication & Authorization
- Using Docker to deploy Alerta

Note: If you require help with any of the above tutorials post a question on [Slack](#).

1.17.2 How-to Guides

How To Monitor Nagios Alerts with Alerta on Ubuntu 16.04 by Vadym Kalsin

How To Monitor Zabbix Alerts with Alerta on CentOS 7 by Vadym Kalsin

OpenSource Metric Based Monitoring by Christian Eichelmann

Installing Alerta on Debian | Ubuntu in Cyber Defence Monitoring Course Suite (CDMCS)

SRE Engineering Practice – Alarm Based on Time Series to Store Data on Docker Mail

Simple tutorial for wetting your appetite on using alerta.io by deeplook

1.18 Resources

1.18.1 Webinars & Slides

How to avoid failing at failure detection by Alex Tavgen, Technical Architect at Playtech

Winning the metrics battle (finally) [Slides] at Velocity Europe 2012

1.18.2 Articles

Winning the metrics battle by Simon Hildrew and Nick Satterly, The Guardian

Never fail twice by Alex Tavgen

Make better use of Prometheus with Grafana, Telegraf, and Alerta [\$] by Martin Loschwitz, Linux Magazin [DE]

Grafana, Telegraf, Alerta – Prometheus besser nutzen (in German) [\$] by Martin Loschwitz, Linux Magazin [DE]

Riemann Learnings by Antonio Terreno, CTO The Labrador

1.18.3 Papers

Frankenstack: Toward Real-time Red Team Feedback by Markus Kont, Mauno Pihelgas, Bernhards Blumbergs of NATO Cooperative Cyber Defence Centre of Excellence and Kaie Maennel and Toomas Lepik of the Tallinn University of Technology at 2017 IEEE Military Communications Conference

EVE and ADAM: Situation Awareness Tools for NATO CCDCOE Cyber Exercises by Francisco Jesús Rubio Melón of Ingeniería de Sistemas para la Defensa de España, Teemu Uolevi Väisänen of VTT Technical Research Centre of Finland and Mauno Pihelgas of NATO Cooperative Cyber Defence Centre of Excellence

1.18.4 References

Event Correlation Engine [Master's Thesis] by Andreas Müller (2009) at Institut für Technische Informatik und Kommunikationsnetze

ANSI/ISA 18.2 Management of Alarm Systems for the Process Industries by American National Standards Institute

1.19 API Reference

Resource Types

- *Alerts*
 - *Create an alert*
 - *Retrieve an alert*
 - *Set alert status*
 - *Action alert*
 - *Tag and untag alerts*
 - *Update alert attributes*
 - *Add an alert note*
 - *Delete an alert*
 - *Search alerts*
 - *List all alert history*
 - *Get severity and status counts for alerts*
 - *Top 10 alerts by resource*
- *Environments*
 - *List all environments*
- *Services*
 - *List all services*
- *Tags*
 - *List all tags*
- *Blackout Periods*
 - *Create a blackout*
 - *List all blackouts*
 - *Delete a blackout*
- *Heartbeats*
 - *Send a heartbeat*
 - *Get a heartbeat*
 - *List all heartbeats*
 - *Delete a heartbeat*
- *API Keys*
 - *Create an API key*
 - *List all API keys*
 - *Delete an API key*

- *Users*
 - *Create a user*
 - *Update a user*
 - *Update me*
 - *Update user attributes*
 - *Update user me attributes*
 - *List all users*
 - *Delete a user*
- *Permissions*
 - *Create permission*
 - *List all permissions*
 - *Delete a permission*
- *Customers*
 - *Create a customer*
 - *List all customers*
 - *Delete a customer*
- *Management*
 - *Manifest*
 - *Properties*
 - *“Good-to-go” Healthcheck*
 - *“Light” Healthcheck*
 - *“Deep” Healthcheck*
 - *JSON Metrics*
 - *Prometheus Metrics*

Note: All datetime parameters must be in ISO 8601 format in UTC time (using time zone designator “Z”) and expressed to millisecond precision as recommended by the [W3C Date and Time Formats Note](#) eg. 2017-06-19T11:16:19.744Z

1.19.1 Alerts

Create an alert

Creates a new alert, or updates an existing alert if the `environment- resource-event` combination already exists.

```
POST /alert
```

Input

Name	Type	Description
resource	string	Required resource under alarm
event	string	Required event name
environment	string	environment, used to namespace the resource
severity	string	see severity_table table
correlate	list	list of related event names
status	string	see status_table table
service	list	list of effected services
group	string	used to group events of similar type
value	string	event value
text	string	freeform text description
tags	set	set of tags
attributes	dict	dictionary of key-value pairs
origin	string	monitoring component that generated the alert
type	string	event type
createTime	datetime	time alert was generated at the origin
timeout	integer	seconds before alert is considered stale
rawData	string	unprocessed raw data

Note: Only resource and event are mandatory. The status can be dynamically assigned by the Alerta API based on the severity.

Example Request

```
$ curl -XPOST http://localhost:8080/alert \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
  "attributes": {
    "region": "EU"
  },
  "correlate": [
    "HttpServerError",
    "HttpServerOK"
  ],
  "environment": "Production",
  "event": "HttpServerError",
  "group": "Web",
  "origin": "curl",
  "resource": "web01",
  "service": [
    "example.com"
  ],
  "severity": "major",
  "tags": [
    "dc1"
  ]
}
```

(continues on next page)

(continued from previous page)

```
],  
  "text": "Site is down.",  
  "type": "exceptionAlert",  
  "value": "Bad Gateway (501)"  
}]'
```

Example Response

201 CREATED

```
{  
  "alert": {  
    "attributes": {  
      "flapping": false,  
      "ip": "127.0.0.1",  
      "notify": true,  
      "region": "EU"  
    },  
    "correlate": [  
      "HttpServerError",  
      "HttpServerOK"  
    ],  
    "createTime": "2018-01-27T21:00:12.999Z",  
    "customer": null,  
    "duplicateCount": 0,  
    "environment": "Production",  
    "event": "HttpServerError",  
    "group": "Web",  
    "history": [  
      {  
        "event": "HttpServerError",  
        "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",  
        "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",  
        "severity": "major",  
        "status": null,  
        "text": "Site is down.",  
        "type": "severity",  
        "updateTime": "2018-01-27T21:00:12.999Z",  
        "value": "Bad Gateway (501)"  
      }  
    ],  
    "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",  
    "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",  
    "lastReceiveId": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",  
    "lastReceiveTime": "2018-01-27T21:00:13.070Z",  
    "origin": "curl",  
    "previousSeverity": "indeterminate",  
    "rawData": null,  
    "receiveTime": "2018-01-27T21:00:13.070Z",  
    "repeat": false,  
  }  
}
```

(continues on next page)

(continued from previous page)

```
"resource": "web01",
"service": [
  "example.com"
],
"severity": "major",
"status": "open",
"tags": [
  "dc1"
],
"text": "Site is down.",
"timeout": 86400,
"trendIndication": "moreSevere",
"type": "exceptionAlert",
"value": "Bad Gateway (501)"
},
"id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
"status": "ok"
}
```

Example Response (during blackout period)

```
202 ACCEPTED
```

```
{
  "message": "Suppressed alert during blackout period",
  "id": "1711c57e-5c6a-4c39-881b-9d8d174feafe",
  "status": "ok"
}
```

Retrieve an alert

Retrieves an alert with the given alert ID.

```
GET /alert/:id
```

Example Request

```
$ curl http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258 \
-H 'Authorization: Key demo-key'
```

Example Response

200 OK

```
{
  "alert": {
    "attributes": {
      "flapping": false,
      "ip": "127.0.0.1",
      "notify": true,
      "region": "EU"
    },
    "correlate": [
      "HttpServerError",
      "HttpServerOK"
    ],
    "createTime": "2018-01-27T21:00:12.999Z",
    "customer": null,
    "duplicateCount": 0,
    "environment": "Production",
    "event": "HttpServerError",
    "group": "Web",
    "history": [
      {
        "event": "HttpServerError",
        "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
        "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
        "severity": "major",
        "status": null,
        "text": "Site is down.",
        "type": "severity",
        "updateTime": "2018-01-27T21:00:12.999Z",
        "value": "Bad Gateway (501)"
      }
    ],
    "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
    "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
    "lastReceiveId": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
    "lastReceiveTime": "2018-01-27T21:00:13.070Z",
    "origin": "curl",
    "previousSeverity": "indeterminate",
    "rawData": null,
    "receiveTime": "2018-01-27T21:00:13.070Z",
    "repeat": false,
    "resource": "web01",
    "service": [
      "example.com"
    ],
    "severity": "major",
    "status": "open",
    "tags": [
      "dc1"
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```
],
  "text": "Site is down.",
  "timeout": 86400,
  "trendIndication": "moreSevere",
  "type": "exceptionAlert",
  "value": "Bad Gateway (501)"
},
"status": "ok",
"total": 1
}
```

Set alert status

Sets the status of an alert, and logs the status change to the alert history.

```
PUT /alert/:id/status
```

Input

Name	Type	Description
status	string	Required New status from open, assign, ack, closed, expired
text	string	reason for status change
timeout	integer	Seconds.

Example Request

```
$ curl -XPUT http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258/status \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
  "status": "ack",
  "text": "disk needs replacing.",
  "timeout": 7200
}'
```

Action alert

Takes an action against an alert which can change the status or severity of the alert and logs the action to the alert history.

```
PUT /alert/:id/action
```

Input

Name	Type	Description
action	string	Required Action from ack, unack`shelve, unshelve, close
text	string	reason for action
timeout	integer	Seconds.

Example Request

```
$ curl -XPUT http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258/action \  
-H 'Authorization: Key demo-key' \  
-H 'Content-type: application/json' \  
-d '{  
    "action": "shelve",  
    "text": "swap out servers.",  
    "timeout": 7200  
}'
```

Tag and untag alerts

Adds or removes tag values from the set of tags for an alert.

```
PUT /alert/:id/tag  
PUT /alert/:id/untag
```

Input

Name	Type	Description
tags	set	tags to add or remove

Example Request

```
$ curl -XPUT http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258/tag \  
-H 'Authorization: Key demo-key' \  
-H 'Content-type: application/json' \  
-d '{  
    "tags": [  
        "linux",  
        "linux2.6",  
        "dell"  
    ]  
}'
```

Update alert attributes

Adds, deletes or modifies alert attributes. To delete an attribute assign “null” to the attribute.

```
PUT /alert/:id/attributes
```

Input

Name	Type	Description
attributes	dict	dictionary of key-value attributes

Example Request

```
$ curl -XPUT http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258/attributes \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
  "attributes": {
    "incidentKey": "1234abcd",
    "ip": "10.1.1.1",
    "region": null
  }
}'
```

Add an alert note

Adds a note to an alert.

```
PUT /alert/:id/note
```

Input

Name	Type	Description
note	string	note text

Example Request

```
$ curl -XPUT http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258/note \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{ "note": "This is the sample note" }'
```

Delete an alert

Permanently deletes an alert. It cannot be undone.

```
DELETE /alert/:id
```

Example Request

```
$ curl -XDELETE http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258 \
-H 'Authorization: Key demo-key'
```

Search alerts

Find alerts using various alert attributes or a mongo-type query parameter to filter results.

```
GET /alerts
```

Parameters

Name	Type	Description
<attr>	string	any alert attribute. eg. status=open
q (*)	string	query string <i>query syntax</i> eg. service:Web OR resource:web
from-date	datetime	lastReceiveTime > from-date
to-date	datetime	lastReceiveTime <= to-date (now)
sort-by	string	attr to sort by (default:lastReceiveTime)
reverse	boolean	change direction of default sort order
page	integer	number between 1 and total pages (default: 1)
page-size	integer	default: 1000 (set DEFAULT_PAGE_SIZE)
show-raw-data	boolean	show raw data
show-history	boolean	show alert history

Deprecated since version 6.3: The q parameter using [Mongo-style query](#) format has been replaced with a query format based on [Lucene query syntax](#) supported by both MongoDB and Postgres backends. For more information see [API Query String Syntax](#).

Example Request

```
$ curl http://localhost:8080/alerts?group=Web \
-H 'Authorization: Key demo-key'
```

Example Response

200 OK

```
{
  "alerts": [
    {
      "attributes": {
        "flapping": false,
        "incidentKey": "1234abcd",
        "ip": "10.1.1.1",
        "notify": true
      },
      "correlate": [
        "HttpServerError",
        "HttpServerOK"
      ],
      "createTime": "2018-01-27T21:00:12.999Z",
      "customer": null,
      "duplicateCount": 0,
      "environment": "Production",
      "event": "HttpServerError",
      "group": "Web",
      "history": [
        {
          "event": "HttpServerError",
          "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
          "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
          "severity": "major",
          "status": null,
          "text": "Site is down.",
          "type": "severity",
          "updateTime": "2018-01-27T21:00:12.999Z",
          "value": "Bad Gateway (501)"
        },
        {
          "event": "HttpServerError",
          "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
          "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
          "severity": null,
          "status": "ack",
          "text": "disk needs replacing.",
          "type": "status",
          "updateTime": "2018-01-27T21:04:42.412Z",
          "value": null
        }
      ],
      "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "lastReceiveId": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "lastReceiveTime": "2018-01-27T21:00:13.070Z",
      "origin": "curl",

```

(continues on next page)

(continued from previous page)

```
"previousSeverity": "indeterminate",
"rawData": null,
"receiveTime": "2018-01-27T21:00:13.070Z",
"repeat": false,
"resource": "web01",
"service": [
  "example.com"
],
"severity": "major",
"status": "ack",
"tags": [
  "dc1",
  "linux",
  "linux2.6",
  "dell"
],
"text": "Site is down.",
"timeout": 86400,
"trendIndication": "moreSevere",
"type": "exceptionAlert",
"value": "Bad Gateway (501)"
}
],
"autoRefresh": true,
"lastTime": "2018-01-27T21:00:13.070Z",
"more": false,
"page": 1,
"pageSize": 1000,
"pages": 1,
"severityCounts": {
  "major": 1
},
"status": "ok",
"statusCounts": {
  "ack": 1
},
"total": 1
}
```

List all alert history

Returns a list of alert severity and status changes.

```
GET /alerts/history
```

Parameters

Name	Type	Description
<attr>	string	

Example Request

```
$ curl http://localhost:8080/alerts/history?service=example.com \
-H 'Authorization: Key demo-key'
```

Example Response

200 OK

```
{
  "history": [
    {
      "attributes": {
        "flapping": false,
        "incidentKey": "1234abcd",
        "ip": "10.1.1.1",
        "notify": true
      },
      "customer": null,
      "environment": "Production",
      "event": "HttpServerError",
      "group": "Web",
      "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "origin": "curl",
      "resource": "web01",
      "service": [
        "example.com"
      ],
      "severity": "major",
      "tags": [
        "dcl",
        "linux",
        "linux2.6",
        "dell"
      ],
      "text": "Site is down.",
      "type": "severity",
      "updateTime": "2018-01-27T21:00:12.999Z",
      "value": "Bad Gateway (501)"
    },
    {
      "attributes": {
```

(continues on next page)

(continued from previous page)

```
    "flapping": false,
    "incidentKey": "1234abcd",
    "ip": "10.1.1.1",
    "notify": true
  },
  "customer": null,
  "environment": "Production",
  "event": "HttpServerError",
  "group": "Web",
  "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
  "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
  "origin": "curl",
  "resource": "web01",
  "service": [
    "example.com"
  ],
  "status": "ack",
  "tags": [
    "dc1",
    "linux",
    "linux2.6",
    "dell"
  ],
  "text": "disk needs replacing.",
  "type": "status",
  "updateTime": "2018-01-27T21:04:42.412Z"
}
],
"status": "ok",
"total": 2
}
```

Get severity and status counts for alerts

Returns a count of alerts grouped by severity and status.

```
GET /alerts/count
```


Parameters

Name	Type	Description
<attr>	string	

Example Request

```
$ curl http://localhost:8080/alerts/count?environment=Production \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "severityCounts": {
    "critical": 1,
    "major": 1
  },
  "status": "ok",
  "statusCounts": {
    "ack": 1,
    "open": 1
  },
  "total": 2
}
```

Top 10 alerts by resource

Returns a list of the top 10 resources grouped by an alert attribute. By default it is grouped by `event` but this can be any valid attribute.

```
GET /alerts/top10/count
GET /alerts/top10/flapping
```

Parameters

Name	Type	Description
<attr>	string	
q	dict	mongo query see `Mongo Query Operators` _
group-by	string	any valid alert attribute. Default:event

Example Request

```
$ curl http://localhost:8080/alerts/top10?group-by=group \  
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{  
  "status": "ok",  
  "top10": [  
    {  
      "count": 2,  
      "duplicateCount": 0,  
      "environments": [  
        "Production"  
      ],  
      "group": "Web",  
      "resources": [  
        {  
          "href": "http://localhost:8080/alert/0099bae5-9683-48a1-a49d-f566fe143770",  
          "id": "0099bae5-9683-48a1-a49d-f566fe143770",  
          "resource": "web02"  
        },  
        {  
          "href": "http://localhost:8080/alert/e9fb05a0-b65c-4faa-8868-6f06a74a2b5b",  
          "id": "e9fb05a0-b65c-4faa-8868-6f06a74a2b5b",  
          "resource": "web01"  
        }  
      ],  
      "services": [  
        "example.com"  
      ]  
    },  
    {  
      "count": 1,  
      "duplicateCount": 0,  
      "environments": [  
        "Production"  
      ],  
      "group": "Web",  
      "resources": [  
        {  
          "href": "http://localhost:8080/alert/e9fb05a0-b65c-4faa-8868-6f06a74a2b5b",  
          "id": "e9fb05a0-b65c-4faa-8868-6f06a74a2b5b",  
          "resource": "web01"  
        }  
      ],  
      "services": [  
        "example.com"  
      ]  
    }  
  ],  
  "total": 1  
}
```

1.19.2 Environments

An environment cannot be created – it is a dynamically derived resource based on existing alerts.

List all environments

Returns a list of environments and an alert count for each.

```
GET /environments
```

Parameters

Name	Type	Description
<attr>	string	

Example Request

```
$ curl http://localhost:8080/environments \  
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{  
  "environments": [  
    {  
      "count": 2,  
      "environment": "Production"  
    }  
  ],  
  "status": "ok",  
  "total": 1  
}
```

1.19.3 Services

A service cannot be created – it is a dynamically derived resource based on existing alerts.

List all services

Returns a list of services grouped by environment and an alert count for each.

```
GET /services
```

Parameters

Name	Type	Description
<attr>	string	

Example Request

```
$ curl http://localhost:8080/services?environment=Production \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "services": [
    {
      "count": 2,
      "environment": "Production",
      "service": "example.com"
    }
  ],
  "status": "ok",
  "total": 1
}
```

1.19.4 Tags

A tag cannot be created – it is a dynamically derived resource based on existing alerts.

List all tags

Returns a list of tags grouped by environment and an alert count for each.

```
GET /tags
```

Parameters

Name	Type	Description
<attr>	string	

Example Request

```
$ curl http://localhost:8080/tags?environment=Production \
-H 'Authorization: Key demo-key'
```

Example Response

200 OK

```
{
  "status": "ok",
  "tags": [
    {
      "count": 2,
      "environment": "Production",
      "tag": "linux"
    },
    {
      "count": 1,
      "environment": "Production",
      "tag": "dc2"
    },
    {
      "count": 1,
      "environment": "Production",
      "tag": "hp"
    },
    {
      "count": 2,
      "environment": "Production",
      "tag": "dell"
    },
    {
      "count": 2,
      "environment": "Production",
      "tag": "dc1"
    },
    {
      "count": 2,
      "environment": "Production",
      "tag": "linux2.6"
    }
  ],
}
```

(continues on next page)

(continued from previous page)

```
}  
  "total": 6
```

1.19.5 Blackout Periods

Create a blackout

Create a new blackout period for alert suppression.

```
POST /blackout
```

Input

Name	Type	Description
environment	string	Required
resource	string	
service	list	
event	string	
group	string	
tags	list	
startTime	datetime	start time of blackout. Default: now
endTime	datetime	end time. Default: now + BLACKOUT_DURATION
duration	integer	seconds. Default: BLACKOUT_DURATION Only used if endTime not defined

Example Request

```
$ curl -XPOST http://localhost:8080/blackout \  
-H 'Authorization: Key demo-key' \  
-H 'Content-type: application/json' \  
-d '{  
  "environment": "Production",  
  "service": ["example.com"],  
  "group": "Web"  
}'
```

Example Response

```
201 CREATED
```

```
{  
  "blackout": {  
    "customer": null,  
    "duration": 3600,  
    "endTime": "2018-01-27T22:10:31.705Z",
```

(continues on next page)

(continued from previous page)

```

    "environment": "Production",
    "event": null,
    "group": "Web",
    "href": "http://localhost:8080/blackout/79d12b79-45b9-4419-80e4-1f6c17478eb6",
    "id": "79d12b79-45b9-4419-80e4-1f6c17478eb6",
    "priority": 3,
    "remaining": 3599,
    "resource": null,
    "service": [
        "example.com"
    ],
    "startTime": "2018-01-27T21:10:31.705Z",
    "status": "active",
    "tags": []
  },
  "id": "79d12b79-45b9-4419-80e4-1f6c17478eb6",
  "status": "ok"
}

```

List all blackouts

Returns a list of blackout periods, including expired blackouts.

```
GET /blackouts
```

Example Request

```
$ curl http://localhost:8080/blackouts \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```

{
  "blackouts": [
    {
      "customer": null,
      "duration": 3600,
      "endTime": "2018-01-27T22:10:31.705Z",
      "environment": "Production",
      "event": null,
      "group": "Web",
      "href": "http://localhost:8080/blackout/79d12b79-45b9-4419-80e4-1f6c17478eb6",
      "id": "79d12b79-45b9-4419-80e4-1f6c17478eb6",
      "priority": 3,
      "remaining": 3345,

```

(continues on next page)

(continued from previous page)

```
"resource": null,
"service": [
  "example.com"
],
"startTime": "2018-01-27T21:10:31.705Z",
"status": "active",
"tags": []
},
{
  "customer": null,
  "duration": 3600,
  "endTime": "2018-01-27T22:14:32.377Z",
  "environment": "Development",
  "event": null,
  "group": "Performance",
  "href": "http://localhost:8080/blackout/c17832d4-c477-4eb1-b2d5-662e7a3600be",
  "id": "c17832d4-c477-4eb1-b2d5-662e7a3600be",
  "priority": 5,
  "remaining": 3586,
  "resource": null,
  "service": [],
  "startTime": "2018-01-27T21:14:32.377Z",
  "status": "active",
  "tags": []
}
],
"status": "ok",
"total": 2
}
```

Delete a blackout

Permanently deletes a blackout period. It cannot be undone.

```
DELETE /blackout/:id
```


Example Request

```
$ curl -XDELETE http://localhost:8080/blackout/c17832d4-c477-4eb1-b2d5-662e7a3600be \
-H 'Authorization: Key demo-key'
```

1.19.6 Heartbeats

Send a heartbeat

Creates a new heartbeat, or updates an existing heartbeat if a heartbeat from the `origin` already exists.

```
POST /heartbeat
```

Input

Name	Type	Description
<code>origin</code>	string	
<code>tags</code>	list	
<code>attributes</code>	dict	dictionary of key-value pairs
<code>createTime</code>	datetime	time alert was generated at the origin
<code>timeout</code>	integer	Seconds.

Example Request

```
$ curl -XPOST http://localhost:8080/heartbeat \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
  "origin": "cluster05",
  "timeout": 120,
  "tags": ["db05", "dc2"],
  "attributes": {
    "environment": "Production",
    "service": [
      "Core",
      "HA"
    ],
    "group": "Network",
    "severity": "major"
  }
}'
```

Example Response

```
201 CREATED
```

```
{
  "heartbeat": {
    "attributes": {
      "environment": "Production",
      "group": "Network",
      "service": [
        "Core",
        "HA"
      ],
      "severity": "major"
    },
    "createTime": "2020-06-07T20:31:58.244Z",
    "customer": null,
    "href": "http://localhost:8080/heartbeat/ea2f41e3-16c4-412f-aaf2-874e3c4c771b",
    "id": "ea2f41e3-16c4-412f-aaf2-874e3c4c771b",
    "latency": 0,
    "maxLatency": 2000,
    "origin": "cluster05",
    "receiveTime": "2020-06-07T20:31:58.244Z",
    "since": 0,
    "status": "ok",
    "tags": [
      "db05",
      "dc2"
    ],
    "timeout": 120,
    "type": "Heartbeat"
  },
  "id": "ea2f41e3-16c4-412f-aaf2-874e3c4c771b",
  "status": "ok"
}
```

Get a heartbeat

Retrieves a heartbeat based on the heartbeat ID.

```
GET /heartbeat/:id
```

Example Request

```
$ curl http://localhost:8080/heartbeat/ea2f41e3-16c4-412f-aaf2-874e3c4c771b \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "heartbeat": {
    "attributes": {
      "environment": "Production",
      "group": "Network",
      "service": [
        "Core",
        "HA"
      ],
      "severity": "major"
    },
    "createTime": "2020-06-07T20:31:58.244Z",
    "customer": null,
    "href": "http://localhost:8080/heartbeat/ea2f41e3-16c4-412f-aaf2-874e3c4c771b",
    "id": "ea2f41e3-16c4-412f-aaf2-874e3c4c771b",
    "latency": 0,
    "maxLatency": 2000,
    "origin": "cluster05",
    "receiveTime": "2020-06-07T20:31:58.244Z",
    "since": 91,
    "status": "ok",
    "tags": [
      "db05",
      "dc2"
    ],
    "timeout": 120,
    "type": "Heartbeat"
  },
  "status": "ok",
  "total": 1
}
```

List all heartbeats

Returns a list of all heartbeats.

```
GET /heartbeats
```

Example Request

```
$ curl http://localhost:8080/heartbeats \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "heartbeats": [
    {
      "attributes": {
        "environment": "Production",
        "group": "Network",
        "service": [
          "Core",
          "HA"
        ],
        "severity": "major"
      },
      "createTime": "2020-06-07T20:31:58.244Z",
      "customer": null,
      "href": "http://localhost:8080/heartbeat/ea2f41e3-16c4-412f-aaf2-874e3c4c771b",
      "id": "ea2f41e3-16c4-412f-aaf2-874e3c4c771b",
      "latency": 0,
      "maxLatency": 2000,
      "origin": "cluster05",
      "receiveTime": "2020-06-07T20:31:58.244Z",
      "since": 136,
      "status": "expired",
      "tags": [
        "db05",
        "dc2"
      ],
      "timeout": 120,
      "type": "Heartbeat"
    }
  ],
  "status": "ok",
  "total": 1
}
```

Delete a heartbeat

Permanently deletes a heartbeat. It cannot be undone.

```
DELETE /heartbeat/:id
```

Example Request

```
$ curl -XDELETE http://localhost:8080/heartbeat/e0582765-ee64-4944-8a94-1869a079d81f \
-H 'Authorization: Key demo-key'
```

1.19.7 API Keys

Create an API key

Creates a new API key.

```
POST /key
```

Input

Name	Type	Description
user	string	username
scopes	string	admin, write, or read
text	string	freeform description text
expireTime	string	
customer	string	Admin use only

Example Request

```
$ curl -XPOST http://localhost:8080/key \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
  "user": "admin@alerta.io",
  "scopes": ["write"],
  "text": "API key for external system"
}'
```

Example Response

```
201 CREATED
```

```
{
  "data": {
    "count": 0,
    "customer": null,
    "expireTime": "2019-01-27T22:18:42.245Z",
    "href": "http://localhost:8080/key/_Jwm-qaGa0kBM9R1CyyQn-0qxLtBtij4ToQf6beL",
    "id": "ca931aec-4e56-496f-a8d6-be11d93ddaed",
    "key": "_Jwm-qaGa0kBM9R1CyyQn-0qxLtBtij4ToQf6beL",
    "lastUsedTime": null,
    "scopes": [
      "write"
    ],
    "text": "API key for external system",
    "type": "read-write",
    "user": "admin@alerta.io"
  },
  "key": "_Jwm-qaGa0kBM9R1CyyQn-0qxLtBtij4ToQf6beL",
  "status": "ok"
}
```

List all API keys

Returns a list of API keys.

```
GET /keys
```

Example Request

```
$ curl http://localhost:8080/keys \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "keys": [
    {
      "count": 0,
      "customer": null,
      "expireTime": "2019-01-27T22:18:42.245Z",
      "href": "http://localhost:8080/key/_Jwm-qaGa0kBM9R1CyyQn-0qxLtBtij4ToQf6beL",
      "id": "ca931aec-4e56-496f-a8d6-be11d93ddaed",
```

(continues on next page)

(continued from previous page)

```

    "key": "_Jwm-qaGa0kBM9R1CyyQn-0qxLtBtij4ToQf6beL",
    "lastUsedTime": null,
    "scopes": [
        "write"
    ],
    "text": "API key for external system",
    "type": "read-write",
    "user": "admin@alerta.io"
  },
  {
    "count": 21,
    "customer": null,
    "expireTime": "2019-01-27T19:22:27.120Z",
    "href": "http://localhost:8080/key/demo-key",
    "id": "532c9b59-9e90-40d4-8a3b-887362a79e9c",
    "key": "demo-key",
    "lastUsedTime": "2018-01-27T22:19:04.113Z",
    "scopes": [
        "admin",
        "write",
        "read"
    ],
    "text": "Admin key created by alertad script",
    "type": "read-write",
    "user": "foo@foobar.com"
  }
],
"status": "ok",
"total": 2
}

```

Delete an API key

Permanently deletes an API key. It cannot be undone.

```
DELETE /key/:key
```

Example Request

```
$ curl -XDELETE http://localhost:8080/key/532c9b59-9e90-40d4-8a3b-
↪887362a79e9c08rhJSKrdfQWXqRhvSwJQJRZg9yU0s2Z4VLP4855 \
-H 'Authorization: Key demo-key'
```

1.19.8 Users

Create a user

Creates a new Basic Auth user.

```
POST /auth/signup
```

Input

Name	Type	Description
name	string	
email	string	
password	string	
text	string	

Example Request

```
$ curl -XPOST http://localhost:8080/auth/signup \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
  "name": "Joe Bloggs",
  "email": "joe.bloggs@example.com",
  "password": "secret",
  "text": "demo user"
}'
```

Example Response

```
200 OK
```

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJzdWIiOiI4Y2IwYjYyNC0zY2Q3LTQ1YjktOThhNS01ZGZhYzVmMDE2NmMiLCJybn2xlcYI6WyJ1c2VyI10sImZcyI6Imh0dHA6L
↪c5jpr8YksoJmoZ6KUwsYP5fgwZr-jdA4W3JUCbv1vXU"
}
```


Update a user

Updates the specified user by setting the values of the parameters passed. Any parameters not provided will be left unchanged.

```
PUT /user/:user
```

Input

Name	Type	Description
name	string	
email	string	
password	string	
status	string	
roles	set	set of roles
attributes	dict	dictionary of key-value pairs
text	string	
email_verified	boolean	

Example Request

```
$ curl -XPUT http://localhost:8080/user/0a35bfd8-1175-4cd2-96f6-eda9861fd15d \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
    "password": "p8ssw0rd",
    "text": "test user",
    "email_verified": false
}'
```

Update me

Updates the logged in user by setting the values of the parameters passed. Any parameters not provided will be left unchanged.

It is not allowed to update roles, email_verified status or change the email to one that is already associated with another user.

```
PUT /user/me
```

Input

Name	Type	Description
name	string	
email	string	
password	string	
status	string	
attributes	dict	dictionary of key-value pairs
text	string	

Example Request

```
$ curl -XPUT http://localhost:8080/user/me \  
-H 'Authorization: Key demo-key' \  
-H 'Content-type: application/json' \  
-d '{  
    "password": "p8ssw0rd",  
    "text": "test user me"  
}'
```

Update user attributes

Updates the specified user attributes.

```
PUT /user/:id/attributes
```

Input

Name	Type	Description
attributes	dict	dictionary of key-value pairs

Example Request

```
$ curl -XPUT http://localhost:8080/user/0a35bfd8-1175-4cd2-96f6-eda9861fd15d/attributes \  
-H 'Authorization: Key demo-key' \  
-H 'Content-type: application/json' \  
-d '{  
    "attributes": {  
        "team": "developers"  
    }  
}'
```

Update user me attributes

Updates the logged in user attributes.

```
PUT /user/me/attributes
```

Input

Name	Type	Description
attributes	dict	dictionary of key-value pairs

Example Request

```
$ curl -XPUT http://localhost:8080/user/me/attributes \  
-H 'Authorization: Key demo-key' \  
-H 'Content-type: application/json' \  
-d '{  
    "attributes": {  
        "teams": ["developers", "operations"]  
    }  
}'
```

List all users

Returns a list of users.

```
GET /users
```

Example Request

```
$ curl http://localhost:8080/users \  
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{  
  "domains": [  
    "*",  
  ],  
  "groups": [  
    "*",  
  ],  
  "orgs": [  
    "*",  
  ],  
}
```

(continues on next page)

(continued from previous page)

```
    "}"
  ],
  "status": "ok",
  "time": "2017-01-02T00:24:00.393Z",
  "total": 2,
  "users": [
    {
      "createTime": "2017-01-01T23:49:38.486Z",
      "email_verified": false,
      "id": "b91811e7-52dd-4a8f-adae-b4d5c628d6f8",
      "login": "jane.doe@example.org",
      "name": "Jane Doe",
      "provider": "basic",
      "role": "user",
      "text": "demo user"
    },
    {
      "createTime": "2017-01-02T00:23:24.487Z",
      "email_verified": true,
      "id": "166b41d6-849f-440d-ba30-1a5345d86fb6",
      "login": "joe.bloggs@example.com",
      "name": "Joe Bloggs",
      "provider": "basic",
      "role": "user",
      "text": "demo user"
    }
  ]
}
```

Delete a user

Permanently deletes a user. It cannot be undone.

```
DELETE /user/:user
```

Example Request

```
$ curl -XDELETE http://localhost:8080/user/166b41d6-849f-440d-ba30-1a5345d86fb6 \
-H 'Authorization: Key demo-key'
```

1.19.9 Permissions

Create permission

Creates a new permission lookup. Used to match user groups/roles to scopes.

```
POST /perm
```

Input

Name	Type	Description
scopes	string	
match	regex	

Example Request

```
$ curl -XPOST http://localhost:8080/perm \  
-H 'Authorization: Key demo-key' \  
-H 'Content-type: application/json' \  
-d '{  
    "scopes": ["read", "write", "admin:alerts"],  
    "match": "alerta_ops"  
}'
```

Example Response

```
201 CREATED
```

```
{  
  "id": "40c2daee-1d77-44d5-b62d-e3e446396cef",  
  "permission": {  
    "id": "40c2daee-1d77-44d5-b62d-e3e446396cef",  
    "match": "alerta_ops",  
    "scopes": [  
      "read",  
      "write",  
      "admin:keys"  
    ]  
  },  
  "status": "ok"  
}
```

List all permissions

Returns a list of permissions.

```
GET /perms
```

Example Request

```
$ curl http://localhost:8080/perms \  
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{  
  "permissions": [  
    {  
      "id": "5b726183-019f-4add-b6dc-caba87e873f7",  
      "match": "alerta_ro",  
      "scopes": [  
        "read"  
      ]  
    },  
    {  
      "id": "f4c91af3-5222-4201-9da0-02c40122f4c4",  
      "match": "alerta_rw",  
      "scopes": [  
        "read",  
        "write"  
      ]  
    },  
    {  
      "id": "1f84f919-c07a-4bd1-93b0-26e28871257f",  
      "match": "alerta_ops",  
      "scopes": [  
        "read",  
        "write",  
        "admin:keys"  
      ]  
    }  
  ],  
  "status": "ok",  
  "time": "2017-07-29T21:42:30.500Z",  
  "total": 3  
}
```

Delete a permission

Permanently delete a permission. It cannot be undone.

```
DELETE /perm/:perm
```

Example Request

```
$ curl -XDELETE http://localhost:8080/perm/1f84f919-c07a-4bd1-93b0-26e28871257f \
-H 'Authorization: Key demo-key'
```

1.19.10 Customers

Create a customer

Creates a new customer lookup. Used to match user logins to customers.

```
POST /customer
```

Input

Name	Type	Description
customer	string	
match	regex	

Example Request

```
$ curl -XPOST http://localhost:8080/customer \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
  "customer": "Example Corp.",
  "match": "example.com"
}'
```

Example Response

```
201 CREATED
```

```
{
  "customer": {
    "customer": "Example Corp.",
    "id": "289ca657-ea2c-4775-9e07-cc96844cc717",
    "match": "example.com"
  }
}
```

(continues on next page)

(continued from previous page)

```
} ,  
  "id": "289ca657-ea2c-4775-9e07-cc96844cc717",  
  "status": "ok"  
}
```

List all customers

Returns a list of customers.

```
GET /customers
```

Example Request

```
$ curl http://localhost:8080/customers \  
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{  
  "customers": [  
    {  
      "customer": "Example Corp.",  
      "id": "289ca657-ea2c-4775-9e07-cc96844cc717",  
      "match": "example.com"  
    },  
    {  
      "customer": "Example Org.",  
      "id": "90f4e211-c815-4112-9e1a-6e53de5a59c6",  
      "match": "example.org"  
    }  
  ],  
  "status": "ok",  
  "time": "2017-01-02T01:21:38.958Z",  
  "total": 2  
}
```


Delete a customer

Permanently delete a customer. It cannot be undone.

```
DELETE /customer/:customer
```

Example Request

```
$ curl -XDELETE http://localhost:8080/customer/90f4e211-c815-4112-9e1a-6e53de5a59c6 \
-H 'Authorization: Key demo-key'
```

1.19.11 Management

Manifest

Get build info, including build date, release number and git commit sha.

```
GET /management/manifest
```

Example Request

```
$ curl http://localhost:8080/management/manifest \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "build": "PROD",
  "date": "2021-11-22T23:46:52Z",
  "release": "8.6.2",
  "revision": "ecfe6ff2295ddc1a01be5aaeeef7dd9159fdcf9"
}
```

Properties

Get HTTP request variables, environment variables, and application configuration settings for debug purposes.

```
GET /management/properties
```

Example Request

```
$ curl http://localhost:8080/management/properties \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
wsgi.version: (1, 0)
wsgi.url_scheme: http
wsgi.input: <_io.BufferedReader name=10>
wsgi.errors: <_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'>
wsgi.multithread: True
wsgi.multiprocess: False
wsgi.run_once: False
werkzeug.server.shutdown: <function WSGIRequestHandler.make_environ.<locals>.shutdown_
↳server at 0x11232a430>
werkzeug.socket: <socket.socket fd=10, family=AddressFamily.AF_INET, type=SocketKind.
↳SOCK_STREAM, proto=0, laddr=('127.0.0.1', 8080), raddr=('127.0.0.1', 51203)>
SERVER_SOFTWARE: Werkzeug/2.0.2
REQUEST_METHOD: GET
SCRIPT_NAME:
PATH_INFO: /management/properties
```

“Good-to-go” Healthcheck

The “good-to-go” healthcheck checks the database is alive and returns HTTP status codes 200 or 503.

```
GET /management/gtg
```

Note: This healthcheck can be used as a *READINESS* check because it shows the container is ready to start accepting traffic.

Example Request

```
$ curl http://localhost:8080/management/gtg
```

Example Response

```
200 OK
```

```
OK
```

“Light” Healthcheck

The “underscore” healthcheck simply returns HTTP status code 200 OK if the application is up. It *does not* query the database.

```
GET /_
```

Note: This healthcheck can be used as a *LIVENESS* check because it simply shows the container is running.

Example Request

```
$ curl -XGET http://localhost:8080/_
```

Example Response

```
200 OK
```

```
OK
```

“Deep” Healthcheck

This healthcheck checks that all reported heartbeats are not more than 4 times their timeout value and reports HTTP status codes 200 or 503. It implicitly checks the database is up also.

```
GET /management/healthcheck
```

Example Request

```
$ curl -XGET http://localhost:8080/management/healthcheck
```

Example Response

```
200 OK
```

```
OK
```

JSON Metrics

Get application metrics in JSON format.

```
GET /management/status
```

Example Request

```
$ curl -XGET http://localhost:8080/management/status \  
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{  
  "application": "alerta",  
  "metrics": [  
    {  
      "description": "Total number of alerts in the database",  
      "group": "alerts",  
      "name": "total",  
      "title": "Total alerts",  
      "type": "gauge",  
      "value": 0  
    },  
    {  
      "count": 12,  
      "description": "Total time and number of alert queries",  
      "group": "alerts",  
      "name": "queries",  
      "title": "Alert queries",  
      "totalTime": 1210,  
      "type": "timer"  
    }  
  ],  
  "time": 1637794336233,  
  "uptime": 1321373,  
  "version": "8.6.2"  
}
```

Prometheus Metrics

Get application metrics in prometheus format.

```
GET /management/metrics
```

Example Request

```
$ curl -XGET http://localhost:8080/management/metrics \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
# HELP alerta_alerts_total Total number of alerts in the database
# TYPE alerta_alerts_total gauge
alerta_alerts_total 0
# HELP alerta_alerts_queries Total time and number of alert queries
# TYPE alerta_alerts_queries summary
alerta_alerts_queries_count 12
alerta_alerts_queries_sum 1210
# HELP alerta_alerts_counts Total time and number of count queries
# TYPE alerta_alerts_counts summary
alerta_alerts_counts_count 2
alerta_alerts_counts_sum 73
# HELP alerta_uptime_msecs milliseconds since app has started
# TYPE alerta_uptime_msecs counter
alerta_uptime_msecs 1422377
```

1.20 API Query Syntax

Alerta API supports two types of query syntax.

- standard URL query parameters
- queries based on Lucene query syntax

Queries are supported by the following resource endpoints:

- *alerts*
- *environments*
- *services*
- *blackouts*
- *heartbeats*
- *users*
- *customers*

- oembed

1.20.1 Standard URL Query parameters

Alert attributes can be used as search parameters:

- Any combination of valid alert attributes can be used to narrow down results.
- Search syntax is = (equals), != (not equals), =~ (regex match) and !=~ (regex exclude).
- When searching for alert `id` the query will attempt to match against `id` and `lastReceiveId`. The “short id” (ie. first 8-characters) can be used. eg. `id=ba358336` instead of `id=ba358336-802d-40ee-8ace-bf5fa8529280`.
- Use “dot notation” to query custom attributes. eg. `attributes.city=Berlin`
- Alert `history` is limited to the 100 most recent status or severity changes. (set using `HISTORY_LIMIT`)
- If “customer views” is enabled then the appropriate `customer` filter for that user will be automatically applied.

1.20.2 Query String Syntax

The query string syntax is used by the `q` query string parameter. It is based on the Lucene query string syntax and is described below.

- *Search terms*
- *Field names*
- *Nested Field names*
- *Wildcards*
- *Regular expressions*
- *Ranges*
- *Boolean Operators*
 - *OR (|)*
 - *AND (&)*
 - *NOT (!)*
- *Grouping*
- *Field Grouping*

Search terms

A search term can be a single word:

foo

or:

bar

A search term can also be a phrase, surrounded by double quotes, which searches for all the words in the phrase, in the same order:

```
"foo bar"
```

Field names

When no explicit field name is specified to search on in the query string the default field `text` will be used unless a prefix is specified.

For example, where `status` field contains “ack”:

```
status:ack
```

Where the `group` field contains “Network” or “Performance”:

```
group:(Network OR Performance)
group:(Network Performance)
```

Where the `text` field contains the exact phrase “kernel panic”:

```
text:"kernel panic"
```

Where the custom attribute `region` has any non-null value:

```
_exists_:region
```

Nested Field names

New in version 7.5.

Nested fields can be queried using dot notation (.) that includes the complete path of the field.

For example, where the `region` custom attribute is “EU”:

```
attributes.region:EU
```

Where the `vendor` custom attribute is “cisco” or “juniper”:

```
attributes.vendor:(cisco OR juniper)
```

For custom attributes the underscore (.) shortcut can be used to replace the `attributes` parent field name.

So the above can be more simply expressed as:

```
_.region:EU
_.vendor:(cisco OR juniper)
```

Wildcards

Wildcard searches can be used on individual terms using `?` to replace single characters and `*` to replace one or more characters:

To search for “foo”, “fu”, “bar” or “baz” use:

```
f* ba?
```

To search for “test” or “text” use:

```
te?t
```

Regular expressions

Regular expression patterns can be embedded in the query string by wrapping them in forward-slashes (`/`). Typical examples include:

```
/[mb]oat/
```

and:

```
name:/joh?n(ath[oa]n)/
```

To search for numbered devices beginning with “net”, “netwrk” or “network” use:

```
resource:/net(wo?rk)?[0-9]/
```

Note: Regular expressions are implemented by the database backends so there may be subtle differences between [Postgres POSIX regular expressions](#) and [MongoDB PCRE \\$regex pattern matching](#) in practice.

Ranges

Ranges can be specified for numeric or string fields. Inclusive ranges are specified with square brackets `[min TO max]` and exclusive ranges with curly brackets `{min TO max}`:

```
timeout:[1 TO 86400]
group:{alpha TO zulu}
value:{* TO 300}
value:[500 TO *]
```

Ranges with one side unbounded (using `*`) can use a simplified syntax:

```
value:>500
value:>=500
value:<500
value:<=500
```


Boolean Operators

New in version 7.5.

Boolean logic operators can be used to combine search terms. They are always in uppercase letters or can be replaced with a symbol.

OR (||)

The OR operator is the default when no operator is specified. A search succeeds if either of the terms are found. The || can be used in place of the word OR.

To search for “foo bar” or “baz” use:

```
"foo bar" baz
"foo bar" OR baz
"foo bar" || baz
```

AND (&&)

The AND operator is used to combine two terms when both must match. The symbol && can be used in place of the word AND.

To search for “foo bar” and “baz” use:

```
"foo bar" AND baz
"foo bar" && baz
```

NOT (!)

The NOT operator is used to exclude matches that contain the search term directly following NOT. The symbol ! can be used in place of the word NOT.

To search for “foo bar” but not “baz” use:

```
"foo bar" NOT baz
"foo bar" AND NOT baz
"foo bar" !baz
```

The NOT operator can be used with a single term:

```
NOT "foo bar"
```

Grouping

Multiple terms or clauses **must** be grouped together with parentheses, to form sub-queries:

```
(foo OR bar) AND baz
```

Field Grouping

Parentheses can be used to group multiple clauses to a single field:

```
status:(open OR ack)
text:(full text search)
```

Note: The following are not currently supported: boolean operators (+,-), range queries by date, and range queries based on severity levels.

Note: The following will not be supported: fuzziness, proximity searches, and boosting which are features specific to Lucene and/or Elasticsearch.

1.21 Alert Format

Alerts received and sent by Alerta conform to a common alert format. All components of alerta use this message format and any external systems must produce or consume messages in this format also.

1.21.1 Attributes

The following alert attributes are populated at source:

Attribute	Description
resource	resource under alarm, deliberately not host-centric
event	event name eg. NodeDown, QUEUE:LENGTH:EXCEEDED
environment	effected environment, used to namespace the resource
severity	severity of alert (default <code>normal</code>). see Alert Severities table
correlate	list of related event names
status	status of alert (default <code>open</code>). see Alert Status table
service	list of effected services
group	event group used to group events of similar type
value	event value eg. 100%, Down, PingFail, 55ms, ORA-1664
text	freeform text description
tags	set of tags in any format eg. aTag, aDouble:Tag, a:Triple=Tag
attributes	dictionary of key-value pairs
origin	name of monitoring component that generated the alert
type	alert type eg. snmptrapAlert, syslogAlert, gangliaAlert
createTime	UTC date-time the alert was generated in ISO8601 format
timeout	number of seconds before alert is considered stale
rawData	unprocessed data eg. full syslog message or SNMP trap

Note: Only event and resource are mandatory.

Attention: If the reject plugin is enabled (which it is by default) then alerts must have an `environment` attribute that is one of either `Production` or `Development` and it must define a `service` attribute. For more information on configuring or disabling this plugin see [Plugin Settings](#).

1.21.2 Attributes added when processing alerts

Attribute	Description
<code>id</code>	globally unique random UUID
<code>customer</code>	assigned based on the owner of the API key used when submitting the alert, if “Customer Views” is enabled, or can be set if <code>admin</code> user
<code>duplicateCount</code>	count of the number of times this event has been received for a resource
<code>repeat</code>	if <code>duplicateCount</code> is 0 or the alert status has changed then <code>repeat</code> is <code>False</code> , otherwise it is <code>True</code>
<code>previousSeverity</code>	the previous severity of the same event for this resource. if no event or <code>correlate</code> events exist in the database for this resource then it will be <code>unknown</code>
<code>trendIndicator</code>	based on <code>severity</code> and <code>previousSeverity</code> will be one of <code>moreSevere</code> , <code>lessSevere</code> or <code>noChange</code>
<code>receiveTime</code>	UTC datetime the alert was received by the Alerta server daemon
<code>lastReceiveId</code>	the last alert id received for this event
<code>lastReceiveTime</code>	the last time this alert was received. only different to <code>receiveTime</code> if the alert is a duplicate
<code>updateTime</code>	the last time the alert status changed. used to calculate time remaining until an alert times out
<code>history</code>	whenever an alert changes severity or status then a list of key alert attributes are appended to the history log

1.21.3 Alert Status

Status	Status Code
<code>open</code>	1
<code>assign</code>	2
<code>ack</code>	3
<code>closed</code>	4
<code>expired</code>	5
<code>blackout</code>	6
<code>shelved</code>	7
<code>unknown</code>	9

1.21.4 Alert Severities

The Alarms in Syslog [RFC 5674](#) was referenced when defining alert severities.

Severity	Severity Code	Colour
security	0	Black
critical	1	Red
major	2	Orange
minor	3	Yellow
warning	4	Blue
informational	5	Green
debug	6	Purple
trace	7	Grey
indeterminate	8	Silver
cleared	9	Green
normal	9	Green
ok	9	Green
unknown	10	Grey

1.21.5 History Entries

History log entries can be for either severity or status changes.

Attribute	Description
id	alert id that history log entry relates to
event	event name of alert changing severity or status
severity (*)	new severity of alert changing severity
status (+)	new status of alert changing status
value (*)	event value of alert changing severity
text	text describing reason for severity or status change
type	history type eg. action, status, severity or value change
updateTime	UTC date-time the alert triggering the change was created

Note: The severity and value attributes are only added to the history log for alerts with event changes (See * above). And the status attribute is only added to the history log for alerts with status changes (See + above).

1.21.6 Example

```
{
  "attributes": {
    "flapping": false,
    "ip": "127.0.0.1",
    "notify": true,
    "region": "EU"
  },
  "correlate": [
    "HttpServerError",
```

(continues on next page)

(continued from previous page)

```

    "HttpServerOK"
  ],
  "createTime": "2018-01-27T21:00:12.999Z",
  "customer": null,
  "duplicateCount": 0,
  "environment": "Production",
  "event": "HttpServerError",
  "group": "Web",
  "history": [
    {
      "event": "HttpServerError",
      "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "severity": "major",
      "status": null,
      "text": "Site is down.",
      "type": "severity",
      "updateTime": "2018-01-27T21:00:12.999Z",
      "value": "Bad Gateway (501)"
    }
  ],
  "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
  "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
  "lastReceiveId": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
  "lastReceiveTime": "2018-01-27T21:00:13.070Z",
  "origin": "curl",
  "previousSeverity": "indeterminate",
  "rawData": null,
  "receiveTime": "2018-01-27T21:00:13.070Z",
  "repeat": false,
  "resource": "web01",
  "service": [
    "example.com"
  ],
  "severity": "major",
  "status": "open",
  "tags": [
    "dc1"
  ],
  "text": "Site is down.",
  "timeout": 86400,
  "trendIndication": "moreSevere",
  "type": "exceptionAlert",
  "value": "Bad Gateway (501)"
}

```

1.22 Heartbeat Format

Heartbeats received by Alerta conform to the following format.

1.22.1 Attributes

The following heartbeat attributes are populated at source:

Attribute	Description
id	globally unique random UUID
origin	name of monitoring component that generated the heartbeat
tags	set of tags in any format eg. aTag, aDouble:Tag, a:Triple=Tag
attributes	dictionary of key-value pairs
type	heartbeat type. only Heartbeat is currently supported
createTime	UTC date and time the heartbeat was generated in ISO 8601 format
timeout	number of seconds before heartbeat is considered stale

Note: Only origin is mandatory.

1.22.2 Attributes added when processing heartbeats

Attribute	Description
receiveTime	UTC date and time the heartbeat was received by the Alerta server daemon
customer	assigned based on the owner of the API key used when submitting the heartbeat, if “Customer Views” are enabled

1.22.3 Example

```
{
  "attributes": {
    "environment": "Production",
    "group": "Network",
    "service": [
      "Core",
      "HA"
    ],
    "severity": "major"
  },
  "createTime": "2020-06-07T20:31:58.244Z",
  "customer": null,
  "href": "http://api.alerta.io/heartbeat/ea2f41e3-16c4-412f-aaf2-874e3c4c771b",
  "id": "ea2f41e3-16c4-412f-aaf2-874e3c4c771b",
  "latency": 0,
  "maxLatency": 2000,
  "origin": "cluster05",
  "receiveTime": "2020-06-07T20:31:58.244Z",
```

(continues on next page)

(continued from previous page)

```
"since": 91,  
"status": "ok",  
"tags": [  
    "db05",  
    "dc2"  
],  
"timeout": 120,  
"type": "Heartbeat"  
}
```


CONTRIBUTE

- Core project: <https://github.com/alerta/alerta>
- Web UI: <https://github.com/alerta/alerta-webui>
- Python SDK: <https://github.com/alerta/python-alerta-client>
- Contributions and integrations: <https://github.com/alerta/alerta-contrib>
- Docker container: <https://github.com/alerta/docker-alerta>

- Slack: <https://slack.alerta.dev>
- *Frequently Asked Questions*
- Issue Tracker: <https://github.com/alerta/alerta/issues>

3.1 Frequently Asked Questions

3.1.1 Alerta

Why can't I see any alerts in the web browser?

If you can send and query for alerts using the `alerta` CLI tool this problem is almost certainly related to cross-origin browser errors. Open up the Javascript developer console in your browser of choice and look for **CORS** errors like:

```
XMLHttpRequest cannot load http://api.alerta.io/alerts?status=open.  
No 'Access-Control-Allow-Origin' header is present on the requested  
resource. Origin 'http://web.alerta.io' is therefore not allowed access.
```

To fix this you can either serve the web UI from the **same origin** as the API using a web server to *reverse proxy* the web UI or ensure that the API server **allows the origin** where the web UI is hosted by adding it to the `CORS_ORIGINS` *server configuration* setting.

Why do I need to set an environment and service when they are not mandatory?

Only `resource` and `event` are technically required to ensure that Alerta can process alerts correctly. However, the “out-of-the-box” default server setting for `PLUGINS` has the `reject` plugin enabled. This plugin enforces an alert “policy” of requiring an `environment` attribute of either `Production` or `Development` and a value for the `service` attribute.

This is to encourage good habits early in defining useful alert attributes that can be used to “namespace” alerts (this is what the `environment` attribute is for) and so that the web console can organise by `environment` and filter alerts by `service`.

However, one of the principles of Alerta is not to enforce its view of the world on users so the plugin can be *easily configured*, modified or completely disabled. It's up to you.

Can I define custom severity codes and levels?

Yes, you can now completely change the severity names, severity levels and colours. See [Alerta Web UI](#) for more information.

How can I add a priority to an alert eg. High, Medium, Low?

Use a custom attribute called `priority` when sending alerts to Alerta:

```
$ alerta send ... --attributes priority=high ...
```

Alerts of differing priority could be queried by alerta command-line tool using:

```
$ alerta query --filters attributes.priority=high
```

Using the web console to sort alerts by priority as well as severity would require some development effort.

What's the difference between *ack*, *close* and *delete*?

Alerts are meant to auto-close when a corresponding *normal* or *cleared* alert is received for that event-resource combination. If no *normal* alert exists for a particular event (which may be the case for alerts from log files or SNMP traps, for example) then the alert will be deleted when the timeout period has expired. Alerts timeout after 1 day by default but that is configurable on a per-alert basis.

If, as an operator, you want to remove an event from view then you can either *ack* the alert or DELETE it. If the alert is DELETED a new alert with the same event-resource combination will trigger a new notification email (if configured) whereas an *ack*'ed alert will not.

Why don't you have a plugin or integration for X, where X is whatever you use in your job?

We could spend countless hours writing plugins for everything and never finish or we could focus on building an easily extensible system with great documentation and let the end-user build the plugins they need. Having said that, we have still [created many plugins](#) and [integrations](#) as working examples and we are not against writing more if there is [popular demand](#). We are also happy to accept submissions.

What's this MongoDB “ServerSelectionTimeoutError”?

With the update to PyMongo 3.x [multiprocessing](#) applications “parent process and each child process must create their own instances of MongoClient”.

For Apache WSGI applications, an example Apache “vhost” configuration for the Alerta API would look like this:

```
<VirtualHost *:8080>
  ServerName localhost
  WSGIDaemonProcess alerta processes=5 threads=5
  WSGIProcessGroup alerta
  WSGIApplicationGroup %{GLOBAL}
  WSGIScriptAlias / /var/www/api.wsgi
  WSGIPassAuthorization On
  <Directory /opt/alerta>
    Require all granted
```

(continues on next page)

(continued from previous page)

```
</Directory>  
</VirtualHost>
```

Full examples are available on [GitHub](#) and more information on why this is necessary is available on [stackoverflow](#) and the PyMongo where they discussion PyMongo in relation to [forking](#) and [mod_wsgi](#) site.

Does Alerta support Python 2.7 or Python 3?

Alerta [Release 5.2](#) is the [last version](#) to support Python 2.7 and from 31 August, 2018 it will only receive critical bug fixes and security patches.

Alerta [Release 6](#) supports Python 3.5+ and is recommended for new production environments and existing installations should be switched to Python 3 well before 1 January, 2020 when Python 2.7 becomes [End-of-Life](#).

LICENSE

This project is licensed under the Apache license, Version 2.0 .

4.1 Releases

4.1.1 Roadmap

- Improve documentation esp. tutorials and web UI guides
- Custom alert filters and dashboard views
- Use OpenAPI ([Swagger](#)) to generate client libraries
- Use Celery tasks for bulk API requests (productionalize)

4.1.2 Release History

Release 7.0.0 (14-04-2019)

- Web UI [version 7](#) complete redesign based on [Vue.js](#) and [Vuetify](#)
- Supports any OpenID Connect compliant auth provider
- Add user groups for Basic Auth role and customer assignment
- Edit users, groups, customers, blackouts, permissions and API keys
- Added user preferences eg. custom shelve time, date/time formats and dark mode
- Add operator notes to alerts and supplementary notes to alert actions
- Multi-select alerts and actions on hover
- Minimum MongoDB version is now 3.2

Release 6.8.0 (2-3-2019)

- Prevent invalid actions for a given alert status
- Lock down Python package versions for deterministic builds
- And sundry fixes

Release 6.7.0 (17-1-2019)

- Sundry fixes

Release 6.6.0 (1-1-2019)

- Remove dependency on deprecated Google+ API (#788)

Release 6.5.0 (23-11-2018)

- Add missing single resource endpoints (#763)

Release 6.4.0 (14-11-2018)

- Add audit trail for “admin”, “write” and “auth” requests
- Fix bi-directional prometheus integration (#740)

Release 6.3.0 (21-10-2018)

- Enhance query to use *Lucene query syntax*
- Add “proxy fix” server config option if using SSL terminating proxy

Release 6.2.0 (13-10-2018)

- Make web UI alert list columns user-configurable
- Add “take action” method to plugins for triggering external actions
- Allow admins to sign-up new users even when sign-up is disabled
- Add filtering and auto-refresh to Watch List and Top10 web pages
- Show all possible menu options when authentication not enabled
- Use scopes instead of type aliases when defining API key permissions

Release 6.1.0 (11-10-2018)

- Added bulk API endpoints for background processing (experimental)
- Added alternative alarm model based on ISA 18.2 / IEC 62682 (experimental)
- Allow users to replace “alerta” web navbar logo with company logo
- Sort by “Create Time” for better integration with Prometheus
- Lots more Python 3 type annotations (and some resulting bug fixes)
- Remove redundant duplicate messages from API logging
- Run tests against Python 3.7 & MongoDB 4.0 for forward compatibility checking
- Add date/time formats and audio back to web UI config and tweak severity colors
- Add support for X-API-Key authentication header (for OpenAPI support)

Release 6.0.0 (18-09-2018)

- First release to support Python 3 only
- Add static type checking to build pipeline and start type annotations
- Add audit info for blackouts including user and reason
- Support every combination of alert attribute for blackouts
- Config API endpoint for dynamically updating client configuration
- Improved email confirmation and user reset of forgotten passwords

Release 5.2.0 (25-04-2018)

- First release to support Python 3.6+ only
- Final release to support Python 2.7
- LDAP authentication support for BasicAuth logins
- Change “status” endpoints to “action” endpoints
- Allow admin to override customer assigned to an alert

Release 5.1.0 (08-04-2018)

- alarm shelving for temporarily removing alerts from the main alert list
- new blackout status that don’t trigger plugins to keep track of suppressed alerts
- add history entry for de-duplicated alerts with a value change
- multiple customers for auth providers that allow membership of more than one group
- Python 3 support only (no breaking changes for Python 2, yet)

Release 5.0.0 (07-10-2017)

- Support for PostgreSQL (including Amazon RDS and Google Cloud SQL)
- API responses are Gzipped to make everything faster
- Development command line has changed from *alertad* to *alertad run*
- Major code refactor with flatter structure (beware imports! see next)
- WSGI import has changed from *from alerta.app import app* to simply *from alerta import app*
- Plugins import has changed from *from alerta.app import app* to *from alerta.plugins import app*
- Blackout is now a plugin so it can be disabled and replaced with a custom blackout handler
- Switched to using wheels for distribution via PyPI See <http://pythonwheels.com/>
- Alerta API now supports multiple roles for BasicAuth (though not supported in the web UI yet)
- Alert format: *value* is now always cast to a string.
- Added */management/housekeeping* URL to replace *housekeepingAlerts.js* cron job script
- *DATABASE_URL* connection URI setting replaces every other MongoDB setting with a non-mongo specific variable

Release 4.10 (27-07-2017)

- Scope-based permissions model based on [RBAC](#)
- [SAML2](#) authentication user logins
- Prometheus webhook updated to support version 4
- Plugin result chaining for tags and attributes

Release 4.9 (16-03-2017)

- LDAP authentication via [Keycloak](#) support
- [MongoDB SSL](#) connection support
- Pingdom webhook changed to use new “State change” webhook

Release 4.8 (05-09-2016)

- Use GitHub Enterprise for OAuth2 login
- [Riemann](#) webhook integration
- [Telegram](#) webhook and [related plugin](#) for bi-directional integration
- [Grafana](#) webhook integration
- Switch to MongoDB URI connection string format
- Added simple *good-to-go* health check
- Added “flap detection” utility method for use in plugins
- Fix oEmbed API endpoint
- Default severity changed from “unknown” to “indeterminate”

- Add routing rules for plugins

Release 4.7 (24-01-2016)

- [Prometheus](#) webhook integration
- [Google Stackdriver](#) webhook integration
- Configurable severities
- Blackout periods by customer
- Status change hook for plugins
- Require authentication on webhooks if auth enabled
- Limit alert history in MongoDB
- Send email confirmation for Basic Auth sign-ups
- Removed support for Twitter OAuth1

Release 4.6 (26-11-2015)

- Customer views for [multitenancy](#) support
- Authorisation using *Admin* and *User* roles

Release 4.5 (9-9-2015)

- Added ability to blackout alerts for defined periods
- Use GitLab for OAuth2 login
- Python 3 support (both `alerta` client and WSGI server)

Release 4.4 (11-6-2015)

- MongoDB version 3 support

Release 4.3 (12-5-2015)

- Support Basic Auth for user logins

Release 4.2 (13-3-2015)

- PagerDuty webhook integration
- API keys can be *read-only* as well as *read-write*

Release 4.1 (25-2-2015)

- Twitter OAuth login
- API response pagination

Release 4.0 (15-1-2015)

- Change web browser authentication to use JWT tokens
- Improve Google OAuth login and add GitHub OAuth

Release 3.3 (16-12-2014)

- Add Amazon AWS CloudWatch, Pingdom web hook integration
- Slack and HipChat plugins

Release 3.2 (11-10-2014)

- Major refactor and simplification of server architecture
- Add Google OAuth user logins
- API keys for controlling programatic access
- Add support for server-side custom plugins eg. Logstash, AWS SNS, AMQP
- Deprecated RabbitMQ as a dependency

Release 3.1 (9-5-2014)

- Extend API to support new dashboard
- Stability and performance enhancements

Release 3.0 (25-3-2014)

- Deploy server and dashboard as Python WSGI apps
- Add AWS Cloudwatch, PagerDuty and Solarwinds integrations
- Pinger module for host availability checks
- Start development of [version 3](#) console based on AngularJS

Release 2.0 (11-3-2013)

- Major refactoring into python modules and classes
- API rewrite based on Flask microframework
- [Dashboard](#) rewritten using Flask server-side templates
- Integrations for AWS SNS, Syslog, Dynect and URL monitoring

Release 1.0 (27-3-2012)

- CGI script receives alerts and pushes to ActiveMQ message bus
- Background daemon reads message bus, processes and stores to MongoDB
- HTML/JavaScript console displays alerts on web dashboard
- Integrations for AWS EC2, Ganglia, IRC, Kibana, Email and SNMP

4.2 About

Alerta started at [The Guardian](#) out of necessity as a replacement for a [legacy monitoring tool](#) but only after exhaustively evaluating all [credible alternatives](#) first.

Initially all we wanted was to be able to create alert thresholds against the hundreds of thousands of [Ganglia metrics](#) collected for the website and view the alerts in a web console ie. a Ganglia “alerter”. Not having a proper name for this [metrics and monitoring system](#) the working name of “an alerter” stuck and a simple homophone was chosen to aid future Google searches.

In the end, the thresholding of metrics proved very difficult to scale so we eventually split the project in two and metric thresholding was given to Riemann (see [riemann-config](#)) and the alert correlation, de-duplication and visualisation became the “Alerta” project.

Over the years the project has evolved to meet the constantly changing needs of the [Guardian developer teams](#) as they moved to a more agile, dynamic, “[swimlaned](#)” architecture which has meant, for the operations team, a shift from static, self-hosted infrastructure to an internal OpenStack cloud to then finally an external cloud service.

In that time certain key features of Alerta have been deprecated as requirements changed (eg. the message bus, Ganglia, Riemann) and others added (eg. OAuth2 login, CloudWatch, Pingdom, PagerDuty integration). In the process it has been slimmed down to fewer core components making it easier to understand, deploy and manage.

As a result, Alerta is now quite different to the somewhat “over engineered” initial solution but the core concepts of being a flexible, easy-to-use tool remain and it is now a “cloud-ready” solution adapted to the challenges of a fast changing environment.

INDICES AND TABLES

- `genindex`
- `search`

A

- ACTIONS, 38
- ADMIN_ROLES, 32
- ADMIN_USERS, 32, 40
- ALARM_MODEL, 31
- ALERT_TIMEOUT, 37
- ALERTA_API_KEY, 11
- ALERTA_CONF_FILE, 11
- ALERTA_DEFAULT_PROFILE, 11
- ALERTA_ENDPOINT, 11
- ALERTA_SVR_CONF_FILE, 29
- ALLOWED_EMAIL_DOMAINS, 33, 35, 40
- ALLOWED_ENVIRONMENTS, 39, 68
- ALLOWED_GITHUB_ORGS, 35, 40
- ALLOWED_GITLAB_GROUPS, 35, 40
- ALLOWED_KEYCLOAK_ROLES, 35, 41
- ALLOWED_SAML2_GROUPS, 34
- API_KEY_EXPIRE_DAYS, 35
- AUDIT_LOG, 36
- AUDIT_LOG_JSON, 36
- AUDIT_LOG_REDACT, 36
- AUDIT_TRAIL, 36
- AUDIT_URL, 36
- AUTH_PROVIDER, 33, 40
- AUTH_REQUIRED, 32, 40
- AUTO_REFRESH_INTERVAL, 38
- AWS_REGION, 35
- AZURE_TENANT, 34

B

- BASE_URL, 30, 40
- BASIC_AUTH_REALM, 33
- BLACKOUT_ACCEPT, 40
- BLACKOUT_DURATION, 40
- blackouts, 7
- BULK_QUERY_LIMIT, 32

C

- CELERY_BROKER_URL, 32
- CELERY_RESULT_BACKEND, 32
- CLICOLOR, 11
- COGNITO_DOMAIN, 35

- COGNITO_USER_POOL_ID, 35
- COLOR_MAP, 37
- COLUMNS, 38
- CORS_ORIGINS, 36, 41, 127
- CUSTOMER_VIEWS, 32, 40

D

- DATABASE_NAME, 32, 41
- DATABASE_RAISE_ON_ERROR, 32
- DATABASE_URL, 32, 41
- DATE_FORMAT_LONG_DATE, 38
- DATE_FORMAT_MEDIUM_DATE, 38
- DATE_FORMAT_SHORT_TIME, 38
- DEBUG, 11, 30, 40
- DEFAULT_ADMIN_ROLE, 32
- DEFAULT_AUDIO_FILE, 38
- DEFAULT_EXPIRED_DELETE_HRS, 37
- DEFAULT_FIELD, 31
- DEFAULT_INFO_DELETE_HRS, 37
- DEFAULT_NORMAL_SEVERITY, 37
- DEFAULT_PAGE_SIZE, 31
- DEFAULT_PREVIOUS_SEVERITY, 37

E

- EMAIL_VERIFICATION, 37
- environment variable
 - ADMIN_USERS, 40
 - ALERTA_API_KEY, 11
 - ALERTA_CONF_FILE, 11
 - ALERTA_DEFAULT_PROFILE, 11
 - ALERTA_ENDPOINT, 11
 - ALERTA_SVR_CONF_FILE, 29
 - ALLOWED_EMAIL_DOMAINS, 40
 - ALLOWED_ENVIRONMENTS, 68
 - ALLOWED_GITHUB_ORGS, 40
 - ALLOWED_GITLAB_GROUPS, 40
 - ALLOWED_KEYCLOAK_ROLES, 41
 - AUTH_PROVIDER, 40
 - AUTH_REQUIRED, 40
 - BASE_URL, 40
 - CLICOLOR, 11
 - CORS_ORIGINS, 41, 127

- CUSTOMER_VIEWS, 40
- DATABASE_NAME, 41
- DATABASE_URL, 41
- DEBUG, 11, 40
- GITHUB_URL, 40
- GITLAB_URL, 40
- GOOGLE_TRACKING_ID, 41
- KEYCLOAK_REALM, 40
- KEYCLOAK_URL, 40
- MAIL_FROM, 41
- MONGO_PORT, 41
- MONGO_URI, 41
- MONGODB_URI, 41
- MONGOHQ_URL, 41
- MONGOLAB_URI, 41
- OAuth2_CLIENT_ID, 40
- OAuth2_CLIENT_SECRET, 40
- ORIGIN_BLACKLIST, 68
- PINGFEDERATE_OPENID_ACCESS_TOKEN_URL, 41
- PINGFEDERATE_OPENID_PAYLOAD_EMAIL, 41
- PINGFEDERATE_OPENID_PAYLOAD_GROUP, 41
- PINGFEDERATE_OPENID_PAYLOAD_USERNAME, 41
- PINGFEDERATE_PUBKEY_LOCATION, 41
- PINGFEDERATE_TOKEN_ALGORITHM, 41
- PLUGINS, 41
- REQUESTS_CA_BUNDLE, 11
- SECRET_KEY, 40, 42
- SMTP_PASSWORD, 41
- USE_PROXYFIX, 40

G

- GITHUB_URL, 35, 40
- GITLAB_URL, 35, 40
- GOOGLE_TRACKING_ID, 38, 41
- GUEST_DEFAULT_SCOPES, 32

H

- heartbeat
 - stale, 9
- HEARTBEAT_MAX_LATENCY, 37
- HEARTBEAT_TIMEOUT, 37
- HISTORY_LIMIT, 31
- HISTORY_ON_VALUE_CHANGE, 31
- HMAC_AUTH_CREDENTIALS, 35

K

- KEYCLOAK_REALM, 35, 40
- KEYCLOAK_URL, 35, 40

L

- LDAP_DOMAINS, 33
- LDAP_URL, 33
- LOG_BACKUP_COUNT, 30
- LOG_CONFIG_FILE, 30

- LOG_FILE, 30
- LOG_FORMAT, 30
- LOG_HANDLERS, 30
- LOG_MAX_BYTES, 30
- LOG_METHODS, 30

M

- MAIL_FROM, 37, 41
- MAIL_LOCALHOST, 37
- MONGO_PORT, 41
- MONGO_URI, 41
- MONGODB_URI, 41
- MONGOHQ_URL, 41
- MONGOLAB_URI, 41

N

- NOTIFICATION_BLACKOUT, 40

O

- OAuth2_CLIENT_ID, 35, 40
- OAuth2_CLIENT_SECRET, 35, 40
- ORIGIN_BLACKLIST, 39, 68

P

- PINGFEDERATE_OPENID_ACCESS_TOKEN_URL, 41
- PINGFEDERATE_OPENID_PAYLOAD_EMAIL, 41
- PINGFEDERATE_OPENID_PAYLOAD_GROUP, 41
- PINGFEDERATE_OPENID_PAYLOAD_USERNAME, 41
- PINGFEDERATE_PUBKEY_LOCATION, 41
- PINGFEDERATE_TOKEN_ALGORITHM, 41
- PLUGINS, 39, 41
- PLUGINS_RAISE_ON_ERROR, 39

R

- REQUESTS_CA_BUNDLE, 11
- RFC
 - RFC 3164, 24
 - RFC 5424, 24
 - RFC 5674, 120

S

- SAML2_CONFIG, 34
- SAML2_USER_NAME_FORMAT, 34
- SECRET_KEY, 30, 40, 42
- SEVERITY_MAP, 37
- SIGNUP_ENABLED, 33
- SITE_LOGO_URL, 38
- SMTP_HOST, 37
- SMTP_PASSWORD, 37, 41
- SMTP_PORT, 37
- SMTP_STARTTLS, 37
- SMTP_USE_SSL, 37
- SMTP_USERNAME, 37

[SORT_LIST_BY](#), [38](#)
[SSL_CERT_FILE](#), [37](#)
[SSL_KEY_FILE](#), [37](#)
stale
 [heartbeat](#), [9](#)

T

[TOKEN_EXPIRE_DAYS](#), [35](#)

U

[USE_PROXYFIX](#), [30](#), [40](#)
[USER_DEFAULT_SCOPES](#), [32](#)